Universidade de Aveiro Departamento de Engenharia Mecânica
2009

Diogo da Silva e Costa

Sistemas de supervisão e controlo integrados

Universidade de Aveiro Departamento de Engenharia Mecânica
2009

Diogo da Silva e Costa

Sistemas de supervisão e controlo integrados

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Professor Doutor José Paulo Santos, Professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro e do Professor Abílio Borges, Professor assistente convidado do Departamento de Engenharia Mecânica da Universidade de Aveiro

Dedico este trabalho à minha Noiva, Pais e Família pelo incansável apoio.

O júri

Presidente

Prof. Doutor Nelson Amadeu Dias Martins
Professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

Prof. Doutor Rui António da Silva Moreira
Professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

Prof. Doutor Carlos Cardeira
Professor do Instituto Superior Técnico de Lisboa

Prof. Doutor José Paulo Oliveira Santos
Professor auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro

Prof. Abílio Manuel Ribeiro Borges
Professor assistente convidado do Departamento de Engenharia Mecânica da Universidade de Aveiro

Agradecimentos

Todo o trabalho desenvolvido apenas se tornou possível com a ajuda e colaboração dos orientadores Professor Doutor José Paulo Santos e Professor Abílio Borges.

Ao Prof. Doutor José Paulo Santos, pelo acompanhamento contínuo e colaboração.

Ao Prof. Abílio Borges, pela sua contribuição e pelas condições criadas.

Um especial agradecimento aos meus pais, que tornaram possível toda a minha vida académica e pessoal que me trouxe até este ponto. Às minhas irmãs, cunhado e sobrinha por todo apoio dado.

Um especial agradecimento à minha Noiva, Ângela, que sempre esteve presente, sempre me apoiou e proporcionou as melhores condições para a evolução dos trabalhos.

À minha colega Ana Soares pela ajuda prestada.

Ao meu colega Hélder Salvador pela ajuda prestada.

A todos os colegas e amigos que sempre me acompanharam.

A todos eles os meus sinceros agradecimentos.

Palavras-chave

Web Services, *Web*, comunicação, *SCADA*, protocolo, supervisão, controlo, dados, aquisição, estado, integração, *OPC,* interoperabilidade.

Resumo

A crescente necessidade do aumento da produtividade e da qualidade na produção de produtos revolucionou o desenvolvimento industrial, visando a utilização de novas tecnologias na produção, no sentido de aumentar a eficiência e a flexibilidade das máquinas e dos processos industriais. Portanto o recurso a autómatos programáveis (*PLC*) para processos automatizados é um dado adquirido, pela versatilidade e fácil integração dos mesmos.

A automatização contínua potenciou a criação de sistemas de supervisão e controlo que são denominados por sistemas *SCADA (Supervision Control And Data Aquisition).* Estes podem ser sistemas locais ou remotos que permitem a total monitorização de um ou vários recursos ou equipamentos.

Os *Web services* são uma tecnologia em desenvolvimento que permite a comunicação entre sistemas, via *WEB*, um recurso que é cada vez mais uma realidade nas mais variadas áreas, tendo em conta as enormes potencialidades que podem ser aplicadas de acordo com as diferentes necessidades.

Os *Web services* funcionam de acordo com o modelo Cliente-Servidor, uma característica que torna possível a utilização dos *Web services* para realizar tarefas de supervisão e controlo de recursos através da *WEB*.

Neste sentido e para permitir a viabilidade da monitorização e controlo remoto através de *Web services* foi desenvolvido um sistema para aplicações *SCADA*, baseado numa arquitectura descentralizada

No âmbito do presente trabalho foram criados diferentes *Web services* que comunicam com uma aplicação servidora local, através de um sistema de gestão de bases de dados. A aplicação local faz uso de rotinas implementadas para estabelecer a comunicação com vários PLC's.

Os *Web services* desenvolvidos são muito fáceis de integrar, uma vez que permitem uma abstracção das camadas inferiores, podendo ser adaptados a diferentes equipamentos.

Os resultados obtidos mostram que este tipo de arquitectura poderá constituir um potencial acrescido para as mais variadas situações e equipamentos, desde ambientes industriais a ambientes académicos e automação predial. As soluções desenvolvidas podem ser facilmente integradas em qualquer ambiente e adaptadas para possibilitarem a supervisão e controlo.

Keywords Web Services, Web, communication, SCADA, protocol, supervision, control, data, acquisition, status, integration, OPC, interoperability.

Abstract

The ongoing demand for higher productivity and quality in the manufacturing of products has revolutionized the industrial development, thus requiring the use of new technologies to improve the efficiency and flexibility of machinery and industrial processes. Therefore the use of PLC (Programmable Logic Controller) for automated processes is indispensable for it's versatility and easy integration.

The continuous use of automated processes has lead to the creation of supervision and control systems also referred to as SCADA (Supervision Control And Data Acquisition). These systems can be local or remote allowing complete monitoring of one or several resources or equipments.

Web services are a technology that's in development and allow communication between systems through the web. This resource is becoming more of a reality in various areas due to it's enormous potential.

Web services work in a Client-Server format, this characteristic makes it possible to accomplish tasks in supervision and control of resources via Web.

In order to allow the viability of monitoring and control via Web services a decentralized based architecture system was developed for SCADA applications.

In the present work different web services were created, communicating with a local server application by means of a database management system. The local application uses routines implemented to establish communication with several PLC's.

The developed web services are very easy to integrate, since they allow an abstraction of lower layers enabling the adaption to different equipments.

The obtained results show that this type of architecture brings an added potential to the most diverse situations and equipments, from industrial environments to academic environments and automation in buildings. The developed solutions can easily be integrated in any type of environment and can easily be adapted to allow supervision and control.

INDÍCE

Lista de Figuras

Lista de tabelas

Lista de acrónimos

A&E	Alarms and Events
API	Application Programming Interface
COM	Component Object Model
DA	Data Access
DCOM	Distribuited Component Object Model
DCS	Distributed Control System
DX	Data Exchange
HDA	Historical Data Access
HMI	Human-Machine Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
MES	Manufacturing Execution System
OLE	Object Linking and Embedding
OPC	Open Process Control
OSI	Open Systems Interconnection
PLC	Programmable Logic Controller
RPC	Remote Procedure Calls
RTU	Remote Terminal Units
SCADA	Supervisory Control And Data Acquisition
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UA	Unified Architecture
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
URI	Universal Resource Identifier
URL	Universal Resource Locator
WSDL	Web Services Definition Language
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language

1 Introdução

Nos dias que correm, com o crescimento das economias, a produção em grande escala é um dado assumido a necessidade de evitar erros, perdas e de diminuir custos tornou imperativo que a automatização fosse parte comum em todas as áreas da indústria, mas não apenas, sendo normal o recurso a autómatos programáveis (*PLC*).

Quando os processos ou linhas de produção automatizadas são mais complexas, associados a produções em série ou em funcionamento contínuo, para que seja fácil acompanhar ou determinar o estado ou mesmo para identificação de possíveis avarias, é possível recorrer a sistemas de supervisão e controlo.

Os sistemas de supervisão e controlo são denominados por *software SCADA* (Supervision Control And Data Aquisition) e podem ser sistemas locais ou remotos, associados a rede local ou mesmo acessíveis a partir da *Internet* [Costa, 2007].

O recurso a processos automatizados possibilita um maior controlo e conhecimento dos mesmos, abrindo portas para os sistemas de supervisão e controlo (*SCADA*), que possibilitam a supervisão e controlo em tempo real ou com apenas algum atraso. Nas grandes unidades fabris, ETAR´s, instalações eléctricas e mesmo em edifícios é já um dado adquirido o recurso a sistemas *SCADA*, permitindo, em alguns casos, o acesso a dados via *Internet* e a supervisão e controlo local ou remoto.

Na realidade das pequenas e médias empresas e outras instalações com menor poder económico, bem como em habitações, o recurso a sistemas *SCADA* requereria um investimento de certa forma elevado, tendo em conta que só a aquisição de um *software SCADA* representaria um grande custo. Neste caso as aplicações “Freeware” ganham um importante relevo, o que significa que o recurso a uma solução que permita a comunicação com *PLC's* e a possibilidade de ser acessível a partir da *Web* seria o ideal. Os *Web services* em conjunto com uma aplicação local apresentam-se como uma possível e interessante solução.

Os *Web services* permitem maior abstracção do cliente, uma vez que não implica a necessita de conhecer os protocolos de comunicação dos *PLC's,* pois funciona de acordo com o modelo Cliente-Servidor, o que poderá possibilitar a supervisão e controlo de recursos controlados por *PLC's* ou outros equipamentos através da *Web*.

Para viabilizar o acesso, generalizado, a sistemas que permitam a supervisão e controlo local e remota, pretende-se conciliar os *Web services* com aplicações para *Windows* que permitam a supervisão e controlo de alguns *PLC's*. O objectivo é chegar a um sistema do tipo Cliente-Servidor, que consiste numa aplicação local para comunicar com os equipamentos e num conjunto de *Web services* que permita ao Cliente aceder aos dados e comunicar de forma indirecta e quase em tempo real com os equipamentos, de tal forma que torna possível a supervisão e controlo dos mesmos.

1.1 Objectivos genéricos

O presente trabalho tem por objectivo criar uma solução que permita a supervisão e controlo de *PLC's* local e remota, de fácil integração com os sistemas existentes e que possibilite ao utilizador aplicar as funções criadas numa aplicação remota de forma transparente e simples, mediante as suas necessidades. Para atingir todos estes pressupostos são avançadas propostas de *Web services* que permitam aceder a recursos remotos, nomeadamente *PLC's*. Os serviços em causa deverão permitir a supervisão e controlo dos equipamentos, o que vai implicar a necessidade desenvolver uma aplicação local e rotinas de comunicação com diferentes *PLC's*.

Os *Web services* deverão ser facilmente integrados e utilizados em aplicações Cliente ou simplesmente utilizados via *Web*.

1.2 Método

O método utilizado para a realização deste trabalho é definido em seis partes, organizadas pela seguinte ordem cronológica:

a. Estudo de sistemas *SCADA* e interfaces *OPC*.

b. Levantamento dos protocolos a que recorrem os *Web services.*

c. Identificação e caracterização dos recursos que se pretendem controlar e integrar.

d. Levantamento e estudo das interfaces de comunicação dos recursos.

e. Definição da arquitectura *SCADA* a propor, baseada em *Web services*.

f. Implementação de uma plataforma de teste da arquitectura proposta.

2 Enquadramento

2.1 Web services

Os *Web services* são uma tecnologia emergente, sobre a qual muito se tem especulado. Uns apontam-na como o caminho a seguir no desenvolvimento de aplicações distribuídas, enquanto outros vêm apenas mais uma evolução de um conceito antigo.

Sendo aplicações modulares, auto-descritivas, acessíveis através de um *URL* (*Universal Resource Locator*), independentes das plataformas de desenvolvimento e que permitem a interacção entre aplicações sem intervenção humana, os Web services apresentam-se como uma possível solução para os actuais problemas de integração entre aplicações cliente, remotas, e recursos [Lopes, 2004].

As características enumeradas em [Lopes, 2004], ficam a dever-se em grande parte ao facto de se basearem em protocolos normalizados, entre os quais se destacam: *XML, SOAP, WSDL e UDDI*.
Segue-se uma breve descrição das funcionalidades de cada um dos protocolos referidos:

- ***XML (eXtensible Markup Language)*** – Linguagem que serve de base a todos os protocolos utilizados nos *Web services*.
- ***SOAP (Simple Object Access Protocol)*** – Protocolo de comunicação, responsável pela troca de mensagens entre cliente e servidor.
- ***WSDL (Web Services Description Language)*** – Formato *XML* responsável pela descrição das *API (Application Programming Interface)* dos *Web services*.
- ***UDDI (Universal Description, Discovery and Integration)*** – Plataforma independente, baseada na *XML*, com a localização e informação sobre os *Web services*, responsável pela descoberta, divulgação e integração dos *mesmos*.

Um requisito básico de qualquer empresa é produzir/fornecer serviços, portanto será crucial encontrar uma forma de facilitar o consumo e o acesso a determinados serviços e é neste sentido que os *Web services* ganham especial relevo.

Um exemplo de integração de *Web services* é realização de uma compra. Este pedido pode e, em muitos casos, já é feito, via interfaces computacionais. O cliente entra no site, cria o pedido como desejar e confirma a compra. Isto é um serviço *Web*, ou seja, um serviço que está publicado na *Web* para que qualquer pessoa possa fazer uso [Pamplona, 2009].

Os *Web services* foram criados para fornecer serviços na internet, a aplicações cliente. Porém não faz parte do conceito de *Web services* o desenvolvimento de interfaces gráficas para os utilizadores, tendo

em atenção que um *Web services* é como um *"Middleware"* (*Software* intermédio) que será integrado às interfaces gráficas desenvolvidas, não sendo de todo, obrigatório a integração em programas para que se possa fazer uso dos *Web services*, pois podem ser utilizados através de um *browser*.

É comum afirmar-se que os *Web services* são serviços para programadores, o que pode ser considerado uma verdade, pois estes disponibilizam as ferramentas necessárias ao programador para executar uma determinada operação/tarefa.

Os *Web services* são a tecnologia ideal para comunicação automática entre sistemas, existindo a aplicação local, que é aquela que disponibiliza o serviço, e a aplicação cliente, que é a aplicação que acede ao serviço [Pamplona, 2009].

Na tabela seguinte estabelece-se uma relação completa entre a aplicação local do *Web services* e a aplicação cliente.

Tabela 1 - Interacção entre aplicação local e aplicação cliente [Chitnis, 2002].

Aplicação local	**Aplicação Cliente**
Define o serviço que será fornecido (*WSDL*)	Identifica os serviços que necessita
Implementa a funcionalidade por trás do serviço	Localiza o serviço pesquisando num directório de serviços (*UDDI*)
Corre a aplicação a fornecer	Envia o pedido ao servidor (*SOAP*)
Publica o *Web services* num directório de serviços (*UDDI*)	Recebe a resposta do servidor (*SOAP*)
Espera por pedidos de aplicações Clientes (*SOAP*)	

. A comunicação entre os serviços é normalizada possibilitando a independência da plataforma e da linguagem de programação. Por exemplo, um sistema feito em Java, a correr num servidor Linux pode aceder a um serviço feito em .Net a correr num servidor *Windows*.

Para comunicar com o *Web Service* é necessário a implementação do protocolo *SOAP,* responsável pela independência que o *Web services* permite. Actualmente, o protocolo *SOAP*, já está implementado em várias plataformas e em várias linguagens de programação.

O diagrama da Figura 1 exemplifica as mensagens trocadas entre cliente e servidor numa comunicação *SOAP*. Existem duas aplicações a comunicar, uma cliente e uma servidora (sendo os serviços empacotados pelos respectivos *Wrappers*), entre eles apenas corre o protocolo *XML*, seguindo o protocolo *SOAP* sobre *HTTP* [Sumra, 2003].

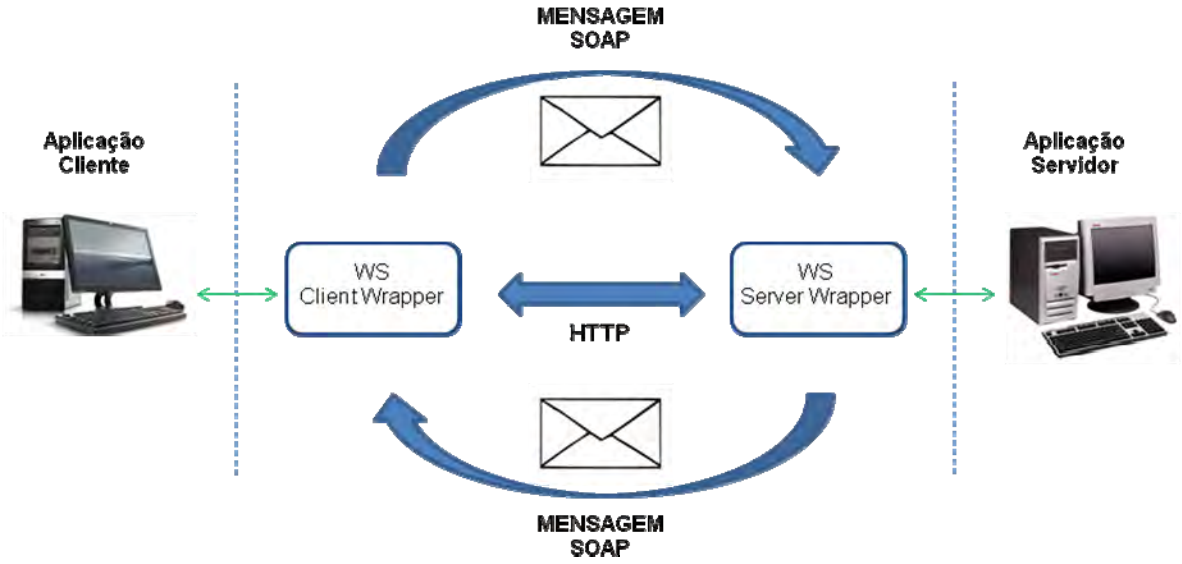


Figura 1 - Representação das mensagens trocadas entre um cliente e um servidor de um *Web Service.*

Para que se possa utilizar um *Web Service* é imperativo definir o modo como se acede ao serviço e que valor devolve. As definições são descritas num arquivo *XML* de acordo com a norma *WSDL. O* arquivo deve ser construído para que os utilizadores do serviço possam entender o funcionamento do *Web services,* sendo de acesso público.

Os *Web services* também podem ser utilizados para implementar arquitecturas orientadas a serviços, as *Service-Oriented Architectures* (*SOA*). Neste modelo de arquitectura os principais requisitos são serviços e são utilizados por outros serviços, modularizando e aumentando a coesão dos componentes da aplicação [Sumra, 2003].

Arquitectura de funcionamento de um *Web Service*

O funcionamento de qualquer *Web services* compreende quatro etapas distintas:
Publicação, Descoberta, Descrição e Invocação. Vejamos com maior pormenor cada etapa [Lopes,2005], [Souza,2006]:

- Publicação: Processo, *opc*ional, através do qual o fornecedor do *Web services* dá a conhecer a existência do seu serviço, efectuando o registo do mesmo no directório de *Web services* (*UDDI*).
- Descoberta: Processo, opcional, através do qual uma aplicação cliente toma conhecimento da existência do *Web services* pretendido.

- Descrição: A aplicação cliente tem acesso a toda a interface do *Web Service*, onde se encontram descritas todas as funcionalidades disponibilizadas, bem como os tipos de mensagens que permitem aceder às ditas funcionalidades.
- Invocação: Processo pelo qual cliente e servidor interagem, através do envio de mensagens de entrada e de eventual recepção de mensagem de saída.

A cada uma destas etapas corresponde um dos protocolos anteriormente referidos.

Ciclo de vida do *Web* Service [Pereira, 2004]:

Um *Web services* apresenta um ciclo de vida característico:

- O fornecedor constrói o serviço utilizando uma linguagem de programação;
- De seguida, descreve o serviço que definiu em WSDL;
- Após a conclusão dos dois primeiros passos, o fornecedor regista o serviço no servidor UDDI;
- O utilizador (aplicação cliente), encontra o serviço procurando-o no servidor UDDI;
- A aplicação cliente estabelece a ligação com o Web services e estabelece um diálogo com este, através de mensagens SOAP.

A referida tecnologia permite que uma aplicação aceda a um ou mais *Web services*, enviando perguntas e recebendo de volta as respostas ou cálculos.

Concluindo, o termo *Web services* descreve uma maneira normalizada de integrar aplicações *Web-based* usando *XML, SOAP, WSDL e UDDI*, por meio de um canal de comunicação sobre o protocolo *IP*. O *XML* é usado para etiquetar os dados, o *SOAP* transfere-os, o *WSDL* descreve os serviços disponíveis e o *UDDI* é usado para listar os fornecedores de *Web services* [Shklar,2003]..

Um *Web service* não fornece uma interface com o utilizador. É da responsabilidade do programador integrar o *Web service* a uma qualquer interface de uma aplicação. Os *Web services* apresentam enormes vantagens, das quais se destacam as seguintes [Inoxnet, 2004], [Lopes,2005]:

- Não requerem instalação nos postos clientes
- Funcionam através de *firewalls*, utilizando a porta 80;
- O cliente não precisa de desenvolver código, sendo suficiente chamar os serviços disponíveis no *Web services*;
- Está acessível em qualquer computador que tenha ligação ao servidor Web;
- São de fácil manutenção/upgrade, pois a aplicação encontra-se apenas no servidor;
- O cliente não precisa de ficar a conhecer a complexidade da implementação do *Web services*;

- O cliente só tem acesso aos serviços disponibilizadas pelo servidor, não tendo acesso directo ao servidor/base de dados;
- São multi-plataformas; uma aplicação Windows pode aceder a um *Web services* a funcionar num sistema *Linux*, por exemplo, e vice-versa.

2.1.1 Situação actual dos *Web services*

Na indústria, os *Web Sevices* fornecem um mecanismo de comunicação entre dois sistemas remotos, ligados através da rede dos *Web services*. Um exemplo é o caso de uma aliança ou aquisição, as empresas não precisam de realizar grandes investimentos para criar um *software* a fim de unir os sistemas das duas empresas. Convertendo e estendendo as aplicações para *Web services*, é possível reunir a informação dos sistemas de duas empresas distintas. Desta forma, é viável aceder aos sistemas empresariais através de simples mensagens *SOAP* sobre o protocolo *HTTP* [Macedo2004].

Um exemplo concreto do uso de *Web services* é o de uma empresa de produção que requer um stock mínimo em armazém de matérias-primas ou material em bruto. Sempre que o stock atinge um valor inferior ao mínimo estabelecido, é necessário repor o *stock.* Esta necessidade poderá ser colmatada com um sistema de controlo do *stock* que o monitoriza e sempre que necessário irá contactar o fornecedor para que o stock seja reposto. O sistema pode ser implementado pelo próprio fornecedor dos materiais, o que permite evitar eventuais disfunções na empresa, para além de deixar de ser necessário que a pessoa supervisora dos *stocks* tome conhecimento da diminuição dos *stocks* e crie uma ordem de compra para os repor, o que pode ser um processo algo moroso [Cunha, 2002].

As potencialidades de aplicação e utilização eficiente dos *Web services* em diversos ramos são inúmeras, dependendo das necessidades de cada empresa, de tal forma que contribui para uma melhor organização e funcionamento das empresas.

O recurso a *Web services* é muito comum na área da economia e finanças, para consultas de bolsa, transacções, intercâmbios, consultas de saldos e de estados de encomendas. Outras áreas de aplicação dos Web *services* são os motores de busca na *Web,* consulta do estado do tempo em cidades ou até mesmo pequenos tradutores ou serviços para envio de *SMS* para as redes móveis.

Com as diversas potencialidades de aplicação, os *Web services* são uma tecnologia muito versátil e que se pode adaptar e ser útil nas mais variadas áreas.

2.2 Sistemas e softwares SCADA

SCADA é a sigla de "*Supervision Control And Data Aquisition*" - Supervisão Controlo e Aquisição de Dados.

SCADA é um termo mais usado para retratar o controlo e a gestão de soluções numa ampla gama de indústrias e pautando-se por um *software* que, num local remoto, permite o acesso a todos os dados de um ou mais processos e gerir os mesmos.

Inicialmente, os sistemas *SCADA* permitiam informar periodicamente o estado do processo industrial, monitorizando sinais representativos de medidas e estados de dispositivos, através de um painel de lâmpadas e indicadores [Wikipedia, 2008].

Com a evolução tecnológica, os computadores assumiram um papel de gestão na recolha e tratamento de dados, tornando possível a sua visualização em computadores e a geração de comandos para execução de funções de controlo complexas

A evolução e optimização dos meios permite que hoje os sistemas *SCADA* utilizem tecnologias de computação e comunicação para automatizar a monitorização e controlo dos processos industriais, efectuando recolha de dados em ambientes industriais, e a respectiva apresentação ao utilizador, com recurso a interfaces Homém-Máquina [Quintas, 2004].

Os sistemas *SCADA* abrangem aplicações cada vez mais diversificadas, podendo estar presentes em diversas áreas, tais como a indústria de celulose, a indústria petrolífera, a indústria têxtil, a indústria metalúrgica, a indústria automóvel, a indústria electrónica, entre outras [Zampronha, 2008].

Estes sistemas revelam-se de crucial importância na estrutura de gestão das empresas, facto pelo qual deixaram de ser vistos como meras ferramentas operacionais e passaram a ser considerados importantes fontes de informação e controlo.

Num ambiente industrial cada vez mais complexo e competitivo, os factores relacionados com a disponibilidade e segurança da informação assumem elevada relevância, tornando-se necessário garantir que a informação está disponível e segura quando necessária, independentemente da localização geográfica. Torna-se portanto necessário implementar mecanismos de acessibilidade, mecanismos de segurança e mecanismos de tolerância a falhas.

Os sistemas *SCADA* melhoram a eficiência do processo de monitorização e controlo, disponibilizando em tempo útil o estado actual do sistema, por meio de um conjunto de previsões, gráficos e relatórios, de modo a permitir a tomada de decisões operacionais apropriadas, quer automaticamente, quer por iniciativa do operador, permitindo desta forma um aumento da segurança nos processos e diminuição de custos. Os benefícios são possíveis através do uso de "*hardware*" e "*software* normalizado" nos sistemas *SCADA* combinados com protocolos de comunicação eficientes e melhoria da conectividade para a rede local e mesmo para a Internet [Quintas, 2004].

SCADA refere-se a um sistema/*software* que recolhe os dados dos vários sensores e actuadores de um ou vários processos em unidades fabris, ou noutros locais remotos e, em seguida, envia os dados para uma unidade central onde são geridos e controlados.

Um *software SCADA* é o topo de uma arquitectura que se desenvolve desde os processos propriamente ditos até ao utilizador/supervisor no seu escritório, casa ou qualquer outro lugar. A esta

arquitectura poder-se-á chamar um sistema *SCADA*, sendo possível ver na Figura seguinte a arquitectura global de um sistema *SCADA*.

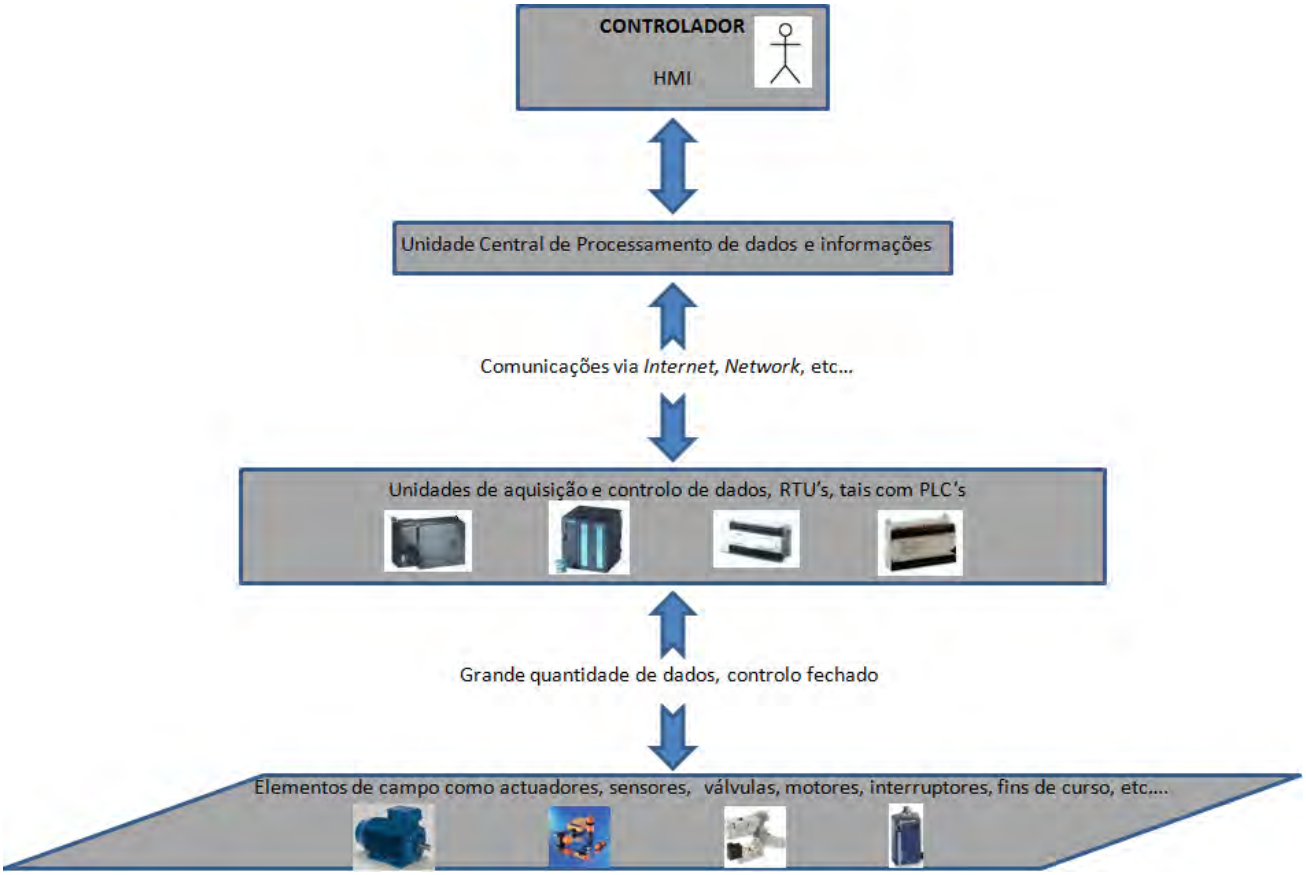


Figura 2 - Arquitectura típica de um sistema *SCADA [Krutz, 2006].*

Um sistema *SCADA* é composto por diversas partes, todas com relevância própria. Normalmente, um sistema *SCADA* inclui *hardware* (de entrada e de saída), controladores, redes, interface com homem (*HMI – Human Machine Interface*), comunicações, equipamentos e *software*. O termo *SCADA* refere-se ao sistema central de dados onde se pode monitorizar e controlar os vários dispositivos do sistema, quer nas proximidades ou fora das instalações.

SCADA pode ser visto como um sistema de dados com muitos elementos chamados pontos. Normalmente cada ponto é um monitor, actuador ou sensor [Krutz, 2006].

Um *software SCADA* monitoriza e controla processos, operações e estados controlados por *RTU*, (*Remote Terminal Units*), na maioria das situações são autómatos.

Os *PLC´s* são definidos de acordo com requisitos específicos e permitem a intervenção humana, por exemplo podem controlar o funcionamento de um transportador, bem como podem alterar a velocidade a qualquer momento. As eventuais alterações ou erros, por norma, são automaticamente detectados e

exibidos. Na maior parte das vezes, um sistema *SCADA* irá acompanhar e fazer ligeiras modificações para que o recurso funcione em óptimas condições; Sistemas *SCADA* são considerados sistemas de circuito fechado e executados com relativamente pouca intervenção humana (Tech-faq, 2008).

Um sistema *SCADA* inclui uma interface para o utilizador/operador, normalmente chamada Interface Homem Máquina, (*HMI*), onde os dados são apresentados para visualização e controlo pelo operador. A referida interface inclui, normalmente, comandos onde o operador pode interagir com o sistema.

As *HMI's* podem ser ligadas a uma base de dados e utilizar as informações recolhidas a partir de *PLC's* ou *RTU's* para proporcionar gráficos, bases de dados sobre sensores, actuadores ou registos de avarias, emergências e possibilitar a criação de históricos. Na última década, praticamente todos os sistemas *SCADA* incluem um dispositivo integrado *HMI-PLC,* o que torna extremamente fácil de executar e monitorar um sistema *SCADA* [Clarke, 2004].

Uma das principais vantagens de um *software SCADA* é a capacidade de controlar todo um sistema em tempo real, algo que é facilitado por aquisições de dados, verificando o estado de sensores, actuadores, etc, sendo comunicados em intervalos regulares, dependendo do sistema. Além dos dados a serem utilizados pelo *RTU/PLC*, também são exibidos ao operador capaz de interagir com o sistema para sobrepor as configurações ou fazer alterações quando necessário.

2.2.1 Estado de arte dos *softwares SCADA*

Necessidade e Vantagens dos *softwares* de supervisão

Nas indústrias, estações de tratamento de água, sistemas de montagem, a qualidade do produto e o tempo que leva para chegar ao mercado são parâmetros chave quando o objectivo é estar em primeiro lugar nos mercados mais competitivos.

As companhias de países mais desenvolvidos, em particular, são forçadas a compensar o preço mais alto da mão-de-obra, sendo que é necessário um processo de produção mais eficiente e transparente, baseado num constante fluxo de informação por todos os níveis e locais da empresa. Actualmente, melhorar o nível de automação, de forma significativa, é bastante difícil e em vez disso passou a ser necessário observar, controlar e tratar os dados recolhidos na empresa para valorizar e optimizar os recursos. Para isso são introduzidos na planta da empresa poderosos sistemas de visualização e modernos sistemas de *SCADA* que recolhem e arquivam dados da produção que se traduzem em importantes pârametros para tomar decisões e que ficam disponíveis de forma a serem facilmente consultados pelos diferentes utilizadores. Admitindo que estes dados estão disponíveis aos operadores, ao departamento de controlo de qualidade, ao departamento de contabilidade, ao departamento de logística e à direcção, todos beneficiam com estas informações [Costa, 2007].

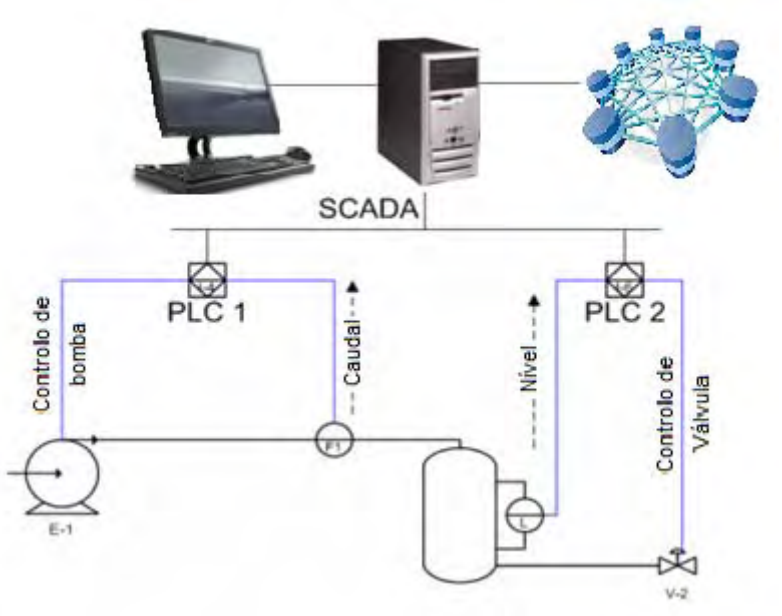


Figura 3 - Exemplo de um sistema *SCADA* [Wikipedia, 2008].

Na Figura 3 vê-se um exemplo prático de aplicação de um *software SCADA*.

A aplicação de um *software SCADA* está em constante evolução, quer em termos gráficos e funcionais, quer na facilidade de programação, no sentido de melhorar a interface com o utilizador e tornar mais atractiva a sua utilização.

As soluções *SCADA* aplicam-se em todas as áreas onde é implementada automação, como por exemplo na gestão técnica de edifícios, infra-estruturas e instalações industriais, em centrais de produção, ETA (Estação de Tratamento de Águas) e ETAR (Estação de Tratamento de Águas Residuais), energia eléctrica, semáforos, sistemas de controlo ambientais, sistemas de produção, linhas de enchimento, entre outras, como em qualquer outro sistema que necessite de controlo e gestão. Os *software*s *SCADA* permitem, de forma eficiente, o controlo e supervisão com dados em tempo real nas operações de produção, possibilitando ainda um acréscimo na segurança para as instalações e pessoas.

Em suma, os *software*s *SCADA* têm como principais vantagens:

- Monitorização remota dos estados de sistemas de automação;
- Capacidade de armazenamento de dados (utilizando bases de dados);
- Capacidade de construção de históricos (gráficos ao longo do tempo, com periodicidade seleccionáveis);
- Ferramentas avançadas de desenho, para realizar Interfaces Homem-Máquina;
- Emissão de alarmes das variáveis supervisionadas;

O Futuro...

No futuro, as indústrias vão continuar a debater-se com o desafio de uma economia global em crescimento. Para sobreviver, crescer e competir, as empresas devem ser capazes de se adaptar o mais rapidamente possível aos seus processos de produção e às constantes mudanças nas condições do mercado. A crescente necessidade de evitar falhas e optimização de processos leva a que a supervisão dos sistemas industriais ganhe contornos cada vez maiores necessitando de evoluir exponencialmente.

2.2.1.1 *Softwares SCADA*

Actualmente, existem alguns *Software*s de Supervisão que se caracterizam ampla componente gráfica e pelos recursos disponíveis, grande parte destes *software*s contêm uma biblioteca de protocolos de comunicação para que possam realizar a supervisão e controlo dos mais variados autómatos existentes. Com esta possibilidade, o trabalho das empresas é facilitado ao adquirirem os *software*s, no entanto, também eleva muito o custo de um *software*, tornando este tipo de produtos inacessíveis às pequenas e médias empresas.

Os *software*s mais conhecidos são:

ZenOn

ZenOn é um poderoso *software SCADA* que permite aos utilizadores do *software* EPLAN, visualizar e controlar partes do sistema sem serem precisos conhecimentos adicionais, utilizando os diagramas eléctricos originais. Através da plataforma do *EPLAN* pode-se aceder directamente ao *ZenOn*, durante o processo de desenvolvimento, da mesma forma o utilizador do *ZenOn* pode aceder à plataforma do *EPLAN*, durante o processamento para detectar alarmes. A poderosa troca de dados entre o desenho da instalação e a manutenção torna a engenharia automática possível [COPA-DATA, 2009].

Figura 4 – Logotipo do Software ZenOn

Recentemente o grupo *BMW,* à semelhança da *FORD,* escolheu o *software ZenOn* para controlar, à escala mundial, as suas empresas de produção.

CX-Supervisor

A *Omron* criou uma plataforma de *software* que permitirá aos fabricantes fazer frente ao desafio do mercado. O *CX Automation Suite* da *Omron,* foi concebido para integrar dinamicamente qualquer implementação, adaptação, melhoria ou qualquer extensão a qualquer tipo de controlo ou aquisição de dados dentro de uma indústria. Este *software* oferece os

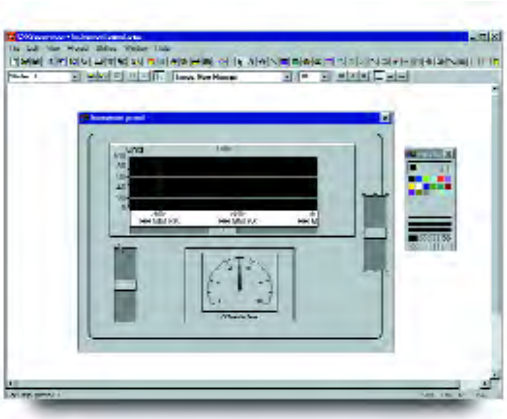


Figura 5 – Exemplo de um projecto em *CX-Supervisor*

requisitos necessários para optimizar de forma constante os processos de produção.

O *CX-Supervisor* é um módulo do *software CX Automation Suite* que permite produzir e executar aplicações para visualização gráfica e controlo de sistemas industriais com *PLC's*. Este módulo foi desenhado e desenvolvido baseando-se nas tecnologias COM/DCOM, OLE, ActiveX, *OPC* e ADO [OMRON, 2008].

A elaboração de aplicações gráficas é uma tarefa simples e rápida. Os objectos gráficos estão disponíveis na barra de ferramentas e podem ser redimensionados e trabalhar facilmente como grafismos do *Microsoft Paint*.

O *CX-Supervisor* permite realizar animações de diferentes elementos de visualização com a mesma facilidade. O editor de animações está especialmente desenhado para manter uma linha de simplicidade. Pode-se seleccionar uma ampla lista de acções que se querem realizar no objecto *Runtime*, e introduzir os nomes das variáveis.

NI-Lookout

Nacional Instruments Lookout é um *software HMI/SCADA* que dá poder e facilidades de utilização para situações inerentes à produção e às aplicações de controlo dos processos.

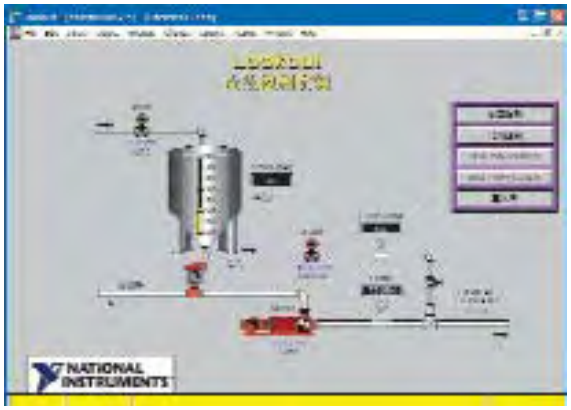


Figura 6 – *Software Lookout*

A informação pode ser recolhida para diminuir o tempo de decisão, enquanto se permite o acesso apenas a pessoas autorizadas. Simplesmente procura-se e selecciona-se a ligação a um determinado computador na rede. O *Lookout* usa uma tecnologia de rede baseada em *TCP/IP* para rapidamente visionar e controlar pontos E/S (Entrada/Saída) em qualquer local da rede. Para além isso pode-se monitorizar e controlar processos usando um *Web browser* [National Instruments, 2009].

WinCC

Simatic WinCC é um processo para visualização de sistemas (*SCADA*) que é dividido por preços e performance, com eficientes funções para controlo de processos automatizados. Com o *Simatic WinCC* são criados processos de visualização perfeitos com funcionalidades completas em controlo e monitorização para todos os segmentos da indústria – permite desde um utilizador único

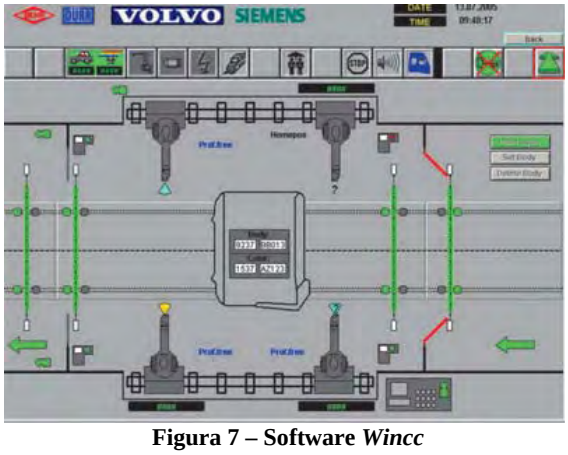


Figura 7 – Software *Wincc*

até uma rede de multi-utilizadores. Este *software* permite, ainda, a criação de interfaces para *HMI's.*

O *WinCC* é um sistema moderno com uma interface atractiva para uso no mundo da produção, oferecendo operações fiáveis e uma configuração eficiente. Em conjunto com o processo integrado de base de dados, o *WinCC* representa a troca de informação dentro da empresa e devido à inteligência de visualização traz uma maior transparência à produção.

Basicamente, o *WinCC* é um *software SCADA* com base de dados em tempo real que é composto por módulos, sendo os principais [Siemens AG, 2007]:

- *WinCC Explorer*: é o gestor dos módulos, utilizado sobretudo para aceder a outros módulos;
- *DM (Data Manager):* através do *Data Manager* configuram-se as *tags* (etiquetas associadas a variáveis), que podem ser internas (sem ligação com o processo) e externas (com ligação com o processo). O processo de configuração das *tags* varia de driver para driver. O *DM* será o responsável por solicitar ao driver as *tags* necessárias para animar uma interface, fazer um gráfico, etc;
- *Alarm Logging*: módulo para gerir os alarmes do sistema;
- *Tag Logging:* módulo para gerir e disparar as rotinas para armazenar as *tags* em períodos/situações previamente configuradas;
- Graphic Designer: editor de telas;
- Global Script: editor de Scripts/Funções.

Movicon

Movicon (Monitoring, Vision, and Control) é um pacote de *software* concebido para criar interfaces homem-máquina, *SCADA*, baseado num *PC* (Personal Computer - computador pessoal).

O *Movicon* permite uma fácil comunicação com o processo com o qual é suposto interagir. Os componentes utilizados na gestão do processo, tais como *PLC's*, controladores de temperatura, etc, constituem o sistema onde o *Movicon* está integrado, comunicando entre si através de portas série, modems, redes *Ethernet*, etc.

O objectivo do *Movicon* é supervisionar os processos de automação, usando interfaces animados, chamados "*synoptic Windows*", ou permitir controlar o processo através da utilização de interfaces denominados "*dialog boxes*", bem como uma variada gama de outras funções que permitem o completo controlo do processo físico de um modo simples e seguro.

Actualmente, a versão *Movicon X* caracteriza-se por uma maior flexibilidade e eficiência. O *Movicon X* pode ser usado por operadores em micro terminais, em *PC's* móveis com *Windows CE*, como também pode ser usado em sistemas distribuídos cliente/servidor, interligado a *PLC's*, redes industriais e redes de campo, em grandes unidades industriais.

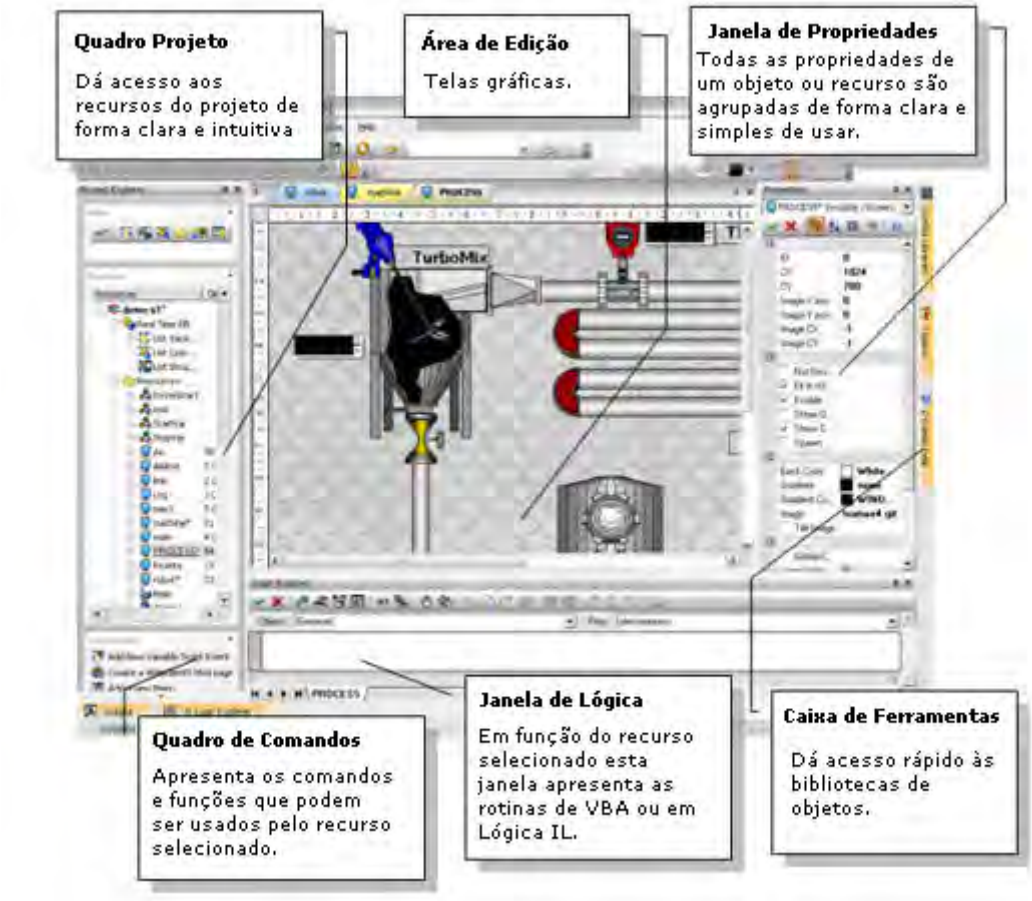


Figura 8 - Editor de projectos do *Movicon X [Exata,2008].*

O *Movicon* X apresenta-se como muito mais simples e intuitivo do que qualquer outro produto, graças a sua área de trabalho inovadora e a ferramentas integradas, tais como: auto-configuração e importação de *tags*.

O *Movicon* X foi construído com tecnologias normalizadas para assegurar seu investimento. Tecnologias tais como: *XML, ODBC, OPC, VBA, SOAP, Web services, TCP-IP, UDP, HTTP e SQL* foram incorporadas ao produto de forma a garantir o acesso fácil e transparente aos dados.

O *software* apresenta ainda uma novidade interessante, a tecnologia *Web Client* que no *Movicon* X surge renovada. Devido à tecnologia *JAVA* que integra perfeitamente com *XML, SVG e Web services*, a nova plataforma permite o acesso ao servidor a partir de "*Browsers* de Internet" em qualquer plataforma (*Windows, Linux, Palm* e alguns sistemas operativos de telemóveis). [Exata,2008], [Quintas, 2004]

A mais recente versão da *Movicon* - o *Movicon 11* - tem uma estrutura baseada em *XML*, completamente direccionada para sistemas do tipo cliente-servidor. O *Movicon 11* tem também módulos baseados na tecnologia *Java* que permitem o acesso a projectos/programas de supervisão e controlo através de um *browser.*

Conclusões

Em suma, os *software*s *SCADA* podem ser divididos em dois tipos, proprietários ou abertos. As empresas desenvolvem *software*s proprietários para comunicar com o seu *hardware* exclusivamente, ou seja, servem apenas para comunicar com os dispositivos desenvolvidos pela mesma empresa.

Os *software*s abertos têm ganho bastante popularidade devido à interoperabilidade a que habilitam os sistemas, o que significa que num mesmo sistema poderão existir dispositivos de diferentes fabricantes e o *software* permitirá a ligação com todos.

De um modo geral, todos os *software*s *SCADA* devem ter as seguintes características:

- Interfaces simples para utilizadores;
- Apresentação de gráficos;
- Gestão e emissão de alarmes;
- Interfaces com os dispositivos (*PLC, RTU*, etc.);
- Escalabilidade;
- Acesso aos dados;
- Bases de dados;
- Possibilidade de integração em rede;
- Processos Cliente/Servidor distribuídos.

2.2.2 *Hardware SCADA*

Como foi possível verificar no ponto 2.2, relativamente à arquitectura de um sistema *SCADA*, este consiste num conjunto de *RTU's* a recolher dados dos dispositivos e a envia-los para uma estação "Mestre", ou estação principal, através do sistema de comunicações implantado.

A estação "Mestre" disponibiliza todos os dados recolhidos dos dispositivos e processos e permite também o controlo remoto dos mesmos.

A comunicação dos dados praticamente em tempo real e com rigor permite a optimização de processos e produção e assim é possível realizar processos de uma forma mais segura [Clarke, 2004].

- Num sistema *SCADA* há essencialmente cinco níveis ou hierarquias;
- Nível de campo, instrumentação e dispositivos de controlo;
- *RTU's* e terminais de triagem e contrlo;
- Sistemas de comunicação;
- Estação(ões) mestre/principal;
- Departamento de processamento de dados e informação tecnológica.

Os *RTU's* fornecem interfaces com sensores analógicos e digitais situados nas várias aplicações/processos.

O sistema de comunicações cria os caminhos necessários para estabelecer comunicações entre a estação "Mestre" e os vários dispositivos remotos. A comunicação pode ser estabelecida de várias formas, por cabos, *wireless*, fibra óptica, linha telefónica, etc. Para optimizar e tornar a transferência de dados são utilizados protocolos específicos e filosofias de detecção de erros.

A estação "Mestre" reúne os dados dos vários *RTU's* e disponibiliza e cria um interface para o operador visualizar a informação e controlar os vários dispositivos remotos.

2.2.3 Redes de comunicação

A tecnologia da informação tem sido determinante no desenvolvimento da tecnologia da automação, alterando hierarquias e estruturas no ambiente dos escritórios até chegar ao ambiente industrial nos seus mais diversos sectores, desde as indústrias até prédios e sistemas logísticos. A capacidade de comunicação entre dispositivos e o uso de mecanismos normalizados, abertos e transparentes, são componentes indispensáveis do conceito de automação de hoje.

Para duas ou mais entidades comunicarem é necessário estabelecer um protocolo de comunicação, definido o formato das mensagens e as regras para a troca de mensagens.

Modelos de alto nível são usados para determinar onde são aplicados os protocolos e definidas as funções necessárias para enviar e receber mensagens. Os modelos de arquitectura por camadas foram amplamente adoptados e mostram-se muito eficazes. Nestes modelos, os elementos necessários para a comunicação são divididos por camadas, com interfaces definidas entre cada camada.

Dois dos modelos de comunicações por camadas mais usados são o modelo *OSI – Open System Interconnection* e o modelo/protocolo *TCP/IP - Transmission Control Protocol/Internet Protocol* [Krutz, 2006].

O modelo *OSI* foi desenvolvido no início dos anos 80 pela *ISO (International Standards Organization)*. Neste modelo os dados de uma camada superior são encapsulados e adicionando um cabeçalho (*header*) e assim transmitidos, e vice-versa de uma camada inferior para uma camada superior [Mackay, 2004]. O modelo *OSI* é constituído por sete camadas, sendo a sétima camada a interface com o utilizador e a primeira a que estabelece a comunicação física com os dispositivos. As camadas do modelo *OSI* são [DEETC, 2008]:

- **Aplicação (7ª)**
  - Oferece serviços de rede às aplicações baseados em protocolos
- **Apresentação (6ª)**
  - Apresentação de dados;
  - Conversões de formatos entre máquinas.
- **Sessão (5ª)**

- Estabelece a comunicação entre a origem e o destino.

- **Transporte (4ª)**
  - Liga processos em computadores diferentes;
  - Cria o conceito de ligação.
- **Rede (3ª)**
  - Fornece o endereço global na rede;
  - Cria o conceito de pacote.
- **Ligação de Dados (2ª)**
  - Agrupa bits para transmissão;
  - Cria o conceito de trama.
- **Física (1ª)**
  - *Hardware* que compõe uma rede;
  - Transforma bits em sinais eléctricos, *wireless*, etc.

Na década de 70 surge o modelo *TCP/IP*, desenvolvido pelo departamento de defesa dos Estados Unidos com o fim de criar uma rede fiável para comunicações. O *TCP/IP* melhorou e reforçou a Internet e todas as suas capacidades são baseadas nos protocolos *TCP/IP*.

O modelo *OSI* foi desenvolvido após o *TCP/IP*, todavia, tentou manter algumas semelhanças. Uma das maiores diferenças entre o *TCP/IP* e o *OSI* é o número de camadas, tendo em conta que o *TCP/IP* é composto apenas por quatro camadas, das quais se desenvolveram as sete camadas do *OSI* [FEUP, 2002].

O modelo *TCP/IP* foi desenhado para suportar comunicação entre redes físicas distintas e com a integração de interfaces *Sockets*, permitiu o desenvolvimento de aplicações e utilitários.

Após o desenvolvimento da tecnologia *TCP/IP*, muitas empresas, tais como a *NASA*, utilizaram este modelo para interligarem as suas redes com o sistema *DARPA,* o que resultou na entidade conhecida por "*ConnectedInternet*", "*DARPA/NSF Internet*" ou simplesmente "Internet", permitindo às instituições trocarem facilmente informação [Mackay, 2004].

A relevância e reputação do modelo *TCP/IP* resultaram da adopção quase universal, bem como a dimensão que a Internet atingiu.

O modelo *TCP/IP* é constituído por apenas quatro camadas apenas, [DEETC, 2005]:

- **Aplicação (4ª)**
  - Programas que fornecem serviços (Ex: *TELNET, FTP, SMTP*);
  - Escolhe o tipo de transporte necessário.
- **Transporte (3ª)**
  - Fornece forma de comunicação entre duas aplicações (ponto a ponto);
  - Controlo de fluxo;

- Controlo de erros;
- Segmentação e reagrupamento das mensagens.

- **Internet (2ª)**

- Fornece comunicação entre duas máquinas;
- Responsável pelo encaminhamento;
- Verifica a validade dos datagramas recebidos;
- Recebe/envia mensagens *ICMP* de controlo e informação e erros;
- Envia datagramas *IP*.

- **Acesso à rede (1ª)**

- Transmite datagramas *IP* e envia-os para uma rede física específica;
- Recebe datagramas IP de uma rede física específica.

O conceito de *internetworking* permite que um sistema se interligue a diversas redes físicas diferentes e as coloque a funcionar como uma unidade coordenada. Cada rede pode ter os seus sistemas de *hardware*, embora estes estejam "escondidos" pela tecnologia Internet. O protocolo *TCP/IP* é desta forma usado para comunicar através de duas redes interligadas.

De um modo geral, a comunicação vem-se expandindo rapidamente em todos os níveis hierárquicos, no sentido horizontal e vertical, desde os níveis inferiores (*field level* - nível de campo, sensores e actuadores) aos níveis de supervisão e controlo.

De acordo com as características da aplicação e do custo máximo a ser atingido, é possível estabelecer uma combinação gradual e hierárquica de diferentes sistemas de comunicação, tais como *Ethernet, CANbus, PROFIBUS, DeviceNet e AS-Interface (ASI)*, oferecendo as condições ideais de redes abertas em processos industriais e permitindo a integração de sistemas *SCADA* [DEETC, 2005].

2.3 Tecnologia OPC

A sigla *OPC,* abreviatura de *OLE* para controlo de processo (*OLE for Controlo Process*), sendo *OLE* a sigla para *Object Linking and Embedding*. *OPC* é o nome original dado para um conjunto de normas e especificações desenvolvidas em 1996 por um grupo de empresas do ramo da automação. As normas criadas especificam a comunicação em tempo real de dados entre dispositivos de controlo de diferentes marcas [Wikipedia, 2008].

Na sequência do lançamento da primeira versão, foi criada a *OPC Foundation,* com o objectivo de uniformizar as normas, princípios de funcionamento e definir especificações comuns. *OPC* é uma série de especificações, a primeira que originalmente foi chamada de *OPC Specification* é agora denominada por *OPC Data Access Specification*, ou simplesmente *OPC Data Access*.

Se no início a *OPC* significava "*OLE for Process Contro*l", actualmente a *OPC Foundation* definiu que *OPC* já não é um acrónimo mas sim uma tecnologia simplesmente conhecida por *OPC* (*Open Process Control*). No cerne desta alteração está, sobretudo, o enorme uso desta tecnologia nas indústrias de processos e produção em série, mas também pelo facto de ser muito comum na produção discreta, o que justifica o facto das normas *OPC* serem muito conhecidas.

Origens e utilidade:

As especificações da *OPC* foram baseadas nas tecnologias *OLE, COM* (*Component Object Model*), e *DCOM* (*Distributed Component Object Model*) desenvolvidas pela *Microsoft* para a família do sistema operativo *Microsoft Windows* [Kepware Technologies, 2008].

A especificação definiu uma configuração normalizada para objectos, interfaces e métodos para uso no controlo de processos e em aplicações de produção automatizadas de forma a facilitar a interoperabilidade.

A tecnologia *OPC* foi desenvolvida para criar uma ponte entre aplicações *Windows* e *hardware* de controlo de processos . As normas definem métodos consistentes para aceder a dados desde os níveis mais baixos da produção. Estes métodos mantêm-se independentemente do tipo e da origem dos dados.

Os OPC servers disponibilizam um método para que diferentes *software*s possam aceder a dados de um dispositivo de controlo de processos, como um *PLC*. Tradicionalmente, sempre que um *software* tinha de aceder a dados de um dispositivo, era necessário desenvolver uma interface ou um driver. O propósito da tecnologia *OPC* é definir uma interface comum desenvolvida uma única vez, e depois reutilizada, quer por um *software SCADA*, *HMI* ou uma aplicação criada.

A partir do momento em que um *OPC* server é desenvolvido para um dispositivo específico, este pode ser reutilizado por qualquer aplicação que consiga funcionar como um *OPC client*. Os *OPC servers* usam a tecnologia *OLE* da *Microsoft* (também conhecida por *COM, Component Object Model*) para comunicar com as aplicações cliente. A tecnologia *COM* normaliza a troca de informação em tempo real entre aplicações de *software* e *hardware* de processo [OPC Task Force, 1998].

Como foi descrito anteriormente, a tecnologia *OPC* baseia-se num conjunto de especificações para permitir a comunicação entre fontes de dados, tais como *PLC's*, controladores, dispositivos de E/S, bases de dados e *HMI's*, ou outro tipo de aplicações cliente.

A Figura 9 ilustra a Arquitectura *OPC*, introduzida pela *OPC Foundation*. De acordo com a arquitectura *OPC*, um dispositivo precisa apenas de um *driver* normalizado para estar acessível, tratando-se do respectivo *OPC server*. Todas a aplicações cliente podem estar ligadas a esse mesmo dispositivo, localmente ou através da rede, de forma remota. Cada *OPC server,* permite mais do que uma ligação, possibilitando as ligações em rede.

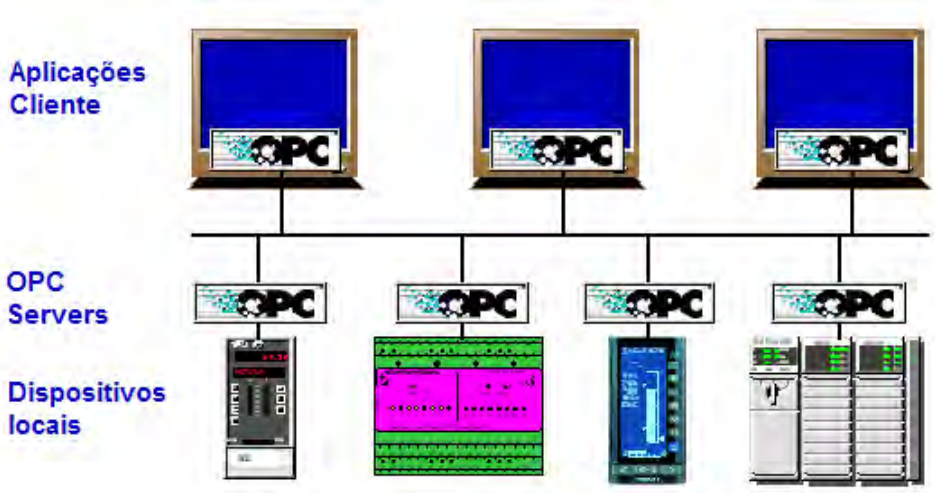


Figura 9 - Arquitectura *OPC*.

A estratégia adoptada passou pela criação de extensões à especificação existente, definição da adição de novas especificações e realização de modificações para manter a compatibilidade máxima com as versões existentes. Em Setembro de 1997 foi definida a primeira actualização da especificação *OPC* que nessa altura passou a ser chamada de *OPC Data Access Specification Version 1.0A* [OPC Task Force, 1998].

Actualmente existem as seguintes especificações publicadas ou em processo de aprovação:

- *OPC Overview* – Descrição geral dos campos de aplicação das especificações *OPC*;
- *OPC Common Definitions and Interfaces* – Definição das funcionalidades básicas para as demais especificações;
- *OPC Data Access Specification* – Definição da interface para leitura e escrita de dados de tempo real;
- *OPC Alarms and Events Specification* – Definição da interface para monitorização de alarmes e eventos;
- *OPC Historical Data Access Specification* – Definição da interface para acesso a dados históricos;
- *OPC Batch Specification* – Definição da interface para acesso aos dados de processos por classe/lotes (*batch*). Esta especificação é uma extensão da *OPC Data Access Specification*;
- *OPC Security Specification*– Definição da interface para utilização de políticas de segurança;
- *OPC XML Data Access Specification* – Integração entre *OPC* e *XML* para aplicações via *Internet (Web*);
- *OPC Unified Architecture* – Uma nova especificação em fase de teste que combina todas as especificações anteriores, para uma maior e melhor interoperabilidade e independência de plataforma.

A Figura 10 apresenta uma visão das especificações da norma *OPC* e respectivas interfaces.

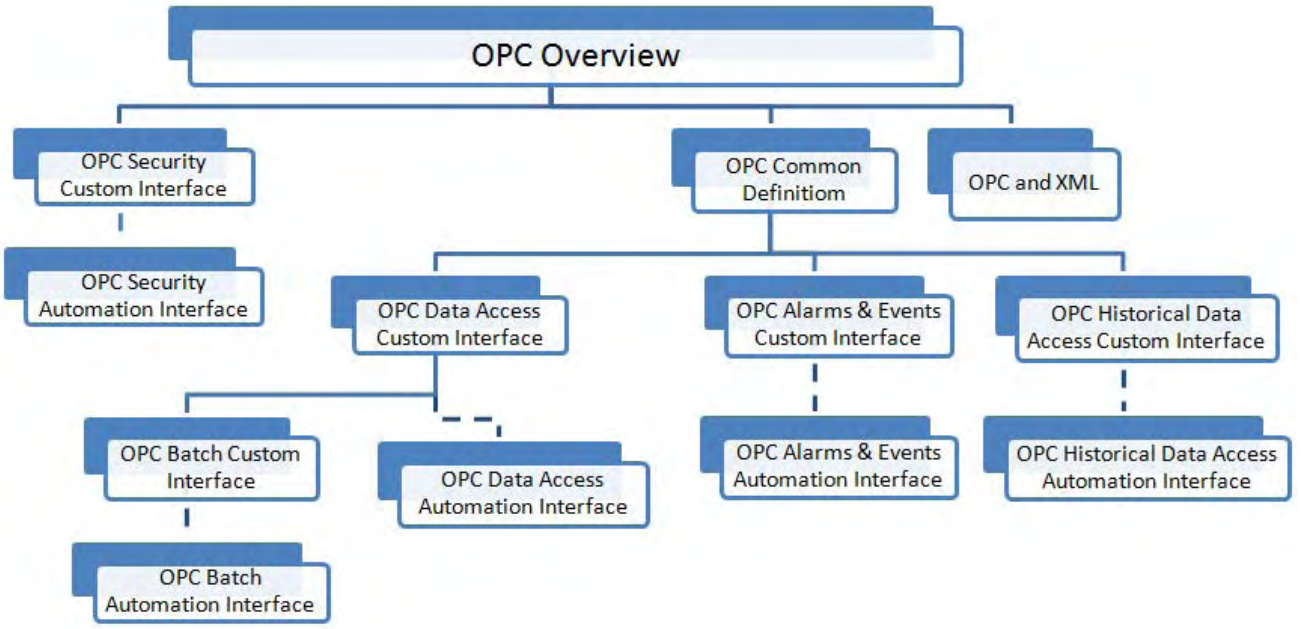


Figura 10 - Especificações da norma *OPC* e respectivas interfaces.

As especificações apresentadas estão em constante desenvolvimento e actualização, sendo que as últimas versões podem ser consultadas no site da *OPC Foundation*. As referidas especificações visam orientar os programadores para a implementação das aplicações cliente e servidor. Em princípio, os utilizadores finais não precisam de conhecer a fundo as especificações, sendo suficiente conhecer os aspectos práticos para utilização da norma [Fonseca, 2002].

Para a implementação de aplicações cliente e servidor, foram desenvolvidas especificações da norma *OPC*, que possibilitam a criação de interfaces para supervisão e controlo de dados de dispositivos.

As interfaces criadas podem ser do tipo *Custom* ou *Automation*, sendo que as primeiras são para aplicações de transferência de elevado volume de dados e de alto rendimento, desenvolvidas em *C/C++, Delphi*, etc. O conceito da interface tipo *Automation* tem por objectivo facultar um acesso mais fácil aos dados, não havendo necessidade de conhecimentos muito elevados de programação para aceder aos dados através de um *OPC Sever.* Existem, portanto, especificações *OPC* separadas para interfaces do tipo *Custom* e *Automation,* em que as interfaces *automation* necessitam de um compilador (*Automation Wrapper*). A Figura 11 apresenta estas interfaces [DeltaV, 2007].

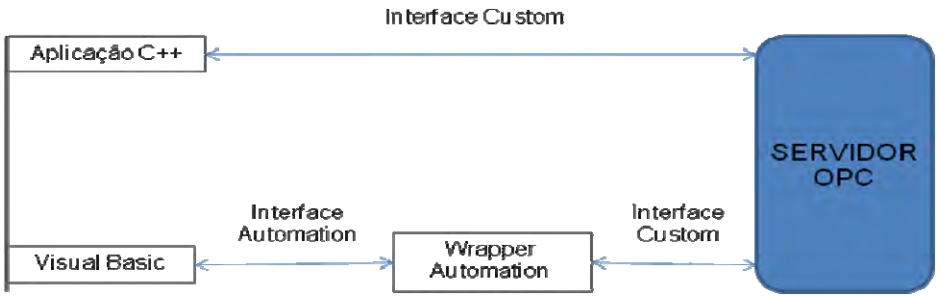


Figura 11 - Interfaces custom e automation.

Para a utilização da norma *OPC*, o utilizador precisa estar ciente de alguns pontos-chave de forma a conseguir beneficiar do uso da comunicação *OPC*. Neste sentido, o estudo das especificações torna-se um processo difícil, uma vez que as mesmas são direccionadas para programadores, o que implica o conhecimento prévio de linguagens e ambientes de desenvolvimento.

Basicamente, a norma *OPC* é mais adaptada para as plataformas *Windows,* existindo variações para as versões do *Windows* (*CE, 9X, NT, 2000, XP* e agora o *Vista*). No caso de plataformas não-*Windows*, há também algumas soluções. No futuro, a especificação *OPC* para *XML* deverá facilitar a integração de plataformas não-*Windows* para a comunicação *OPC*.

Os produtos para monitorização de dados (*HMI*'s; sistemas de supervisão, etc.) são clientes *OPC*, os produtos que fazem a comunicação directa com os dispositivos de campo utilizam protocolos proprietários são servidores *OPC*. Cada produto pode incorporar as duas funcionalidades, sendo mais comum que uma aplicação cliente possa ser servidora, e não o contrário.

O número de servidores *OPC* necessários para uma determinada aplicação depende do produto a ser utilizado. Normalmente, os fabricantes de dispositivos de campo (*PLC*'s; dispositivos inteligentes, etc.) fornecem um servidor *OPC* capaz de utilizar diversos protocolos para comunicar com os seus equipamentos. Este tipo de servidores consiste num *software* para o ambiente *Windows* que é executado num microcomputador, sobretudo um computador pessoal *(PC)*. Por exemplo, o servidor *OPC da Rockwell,* o *RSLinx*, permite que diversos drivers de comunicação sejam configurados para as diversas redes (*ControlNet, DeviceNet, Ethernet, DH+*, etc.). Neste caso, o *RSLinx* funciona como um único servidor *OPC*, capaz de comunicar com diversos clientes *OPC* a ser executados na mesma máquina ou em máquinas remotas [OPC Task Force, 1998].

Desempenho da comunicação *OPC*

Em linhas gerais, o desempenho da comunicação *OPC* aproxima-se do desempenho apresentado por sistemas que utilizam drivers de comunicação específicos e optimizados. Os drivers específicos possuem, normalmente, um óptimo desempenho após serem devidamente depurados e optimizados.

Um servidor *OPC* nada mais é do que uma camada de *software* para implementar as interfaces normalizadas e os mecanismos de comunicação com o cliente, a comunicação é realizada com o cliente e não com o dispositivo de campo.

A utilização de *OPC*'s permite vantagens em relação ao desenvolvimento de projectos onde se utilizem drivers ou soluções proprietárias [Maciel, 2003]:

- Normalização das interfaces de comunicação entre os servidores e clientes de dados em tempo real, facilitando a integração e manutenção dos sistemas;
- Eliminação da necessidade de drivers de comunicação específicos (proprietários);
- Melhoria do desempenho e optimização da comunicação entre dispositivos de automação;
- Interoperabilidade entre sistemas operativos e equipamentos de diversos fabricantes;
- Integração com sistemas *MES, ERP* e aplicações *Windows (Excel*, etc.);
- Redução dos custos e tempo de desenvolvimento de interfaces e drivers de comunicação e integração de sistemas;
- Facilidade de desenvolvimento e manutenção de sistemas e produtos para comunicação em tempo real;
- Uniformidade de interface para diferentes redes e protocolos: com o uso de servidores *OPC* para diferentes redes, o acesso aos dados de processos para supervisão faz-se sem necessidade de ajustes dos drivers de cada rede, não necessitando de configuração;
- Fácil integração com a rede;
- Integração entre diferentes ferramentas de supervisão: o *OPC* disponibiliza um modo simples de integrar diferentes ferramentas de supervisão.

Por outro lado, existem algumas desvantagens na integração de sistemas com *OPC*:

- Velocidade de comunicação: os softwares *OPC* podem ter um desempenho inferior aos que utilizam drivers de comunicação proprietários, pois envolvem um maior número de sistemas;
- Pode existir um maior uso de processamento, pois existem várias aplicações em processamento.
- Manutenção: a manutenção de sistemas que comunicam através de *OPC's* traz um maior número de variáveis para o processo. Falhas nos serviços de comunicação do *Windows*, por exemplo, são frequentemente a origem da maioria das intervenções.

Para este trabalho as especificações que se apresentam com mais relevância são a *OPC Data Access*, a *OPC XML Data Access* e a nova especificação - *OPC Unified Architecture*, estando esta ainda em fase de teste. Actualmente, está em fase de elaboração a especificação *OPC DX Data Exchange* for *Ethernet*, cuja finalidade é definir a comunicação entre diferentes servidores ligados através de rede *Ethernet*

TCP/IP. Até à data, este tipo de comunicação não é possível sem a utilização de um cliente e de uma aplicação para intermediar a troca de dados.

2.3.1 *OPC Data Access*

O *OPC DA* especifica as interfaces de leitura e escrita de dados nos dispositivos de campo, sendo o elo entre a camada de Supervisão e os drivers de comunicação com os dispositivos. A especificação rege as aplicações do tipo Servidor-Cliente (Figura 12), sendo que a aplicação local deverá conter os drivers necessários à comunicação com os dispositivos [DeltaV, 2007].

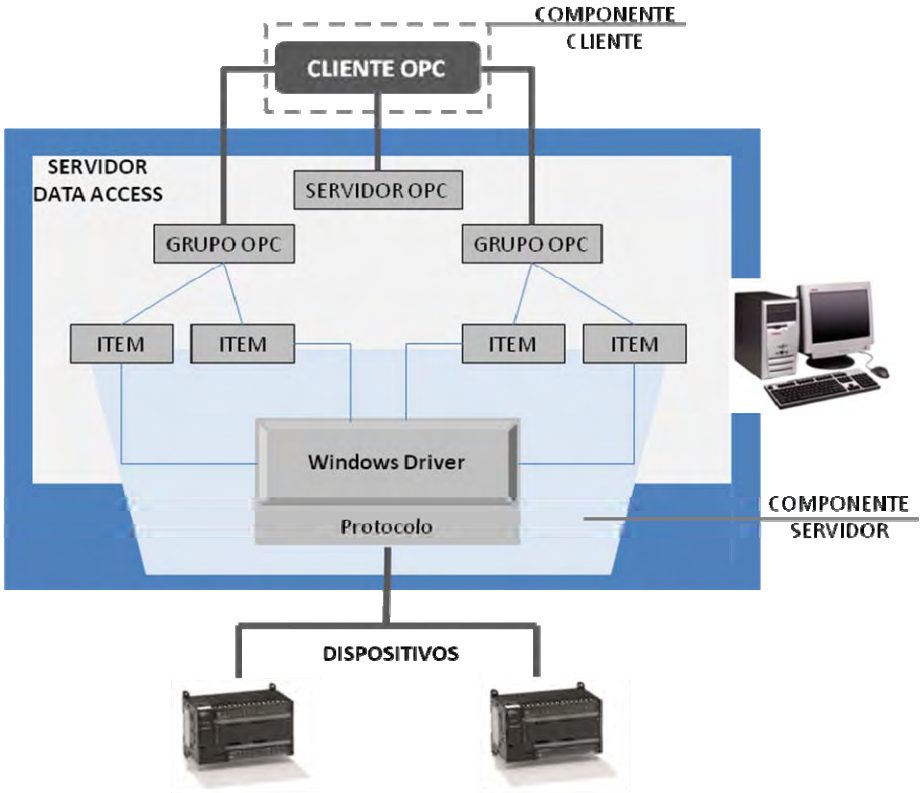


Figura 12 - Relação/arquitectura entre Cliente e Servidor OPC [Fonseca, 2002].

As marcas dos dispositivos, tais como *PLC's*, já têm criados os seus servidores *OPC*, no entanto, estes são proprietários, ou seja, são específicos para um dispositivo ou para uma gama. Existem servidores *OPC,* independentes, que permitem que sejam configurados drivers de comunicação para diversas redes e protocolos de diferentes fabricantes. Exemplos disso são os servidores da *Kepware* e da *Matrikon*. Neste caso, um único produto poderá servir dados de diferentes fabricantes.

Cada cliente *OPC* pode ligar-se a diferentes servidores, a funcionar no mesmo computador ou em computadores diferentes, desta forma, qualquer produto que funcione como cliente *OPC* poderá comunicar com vários servidores *OPC* independentemente do seu fabricante.

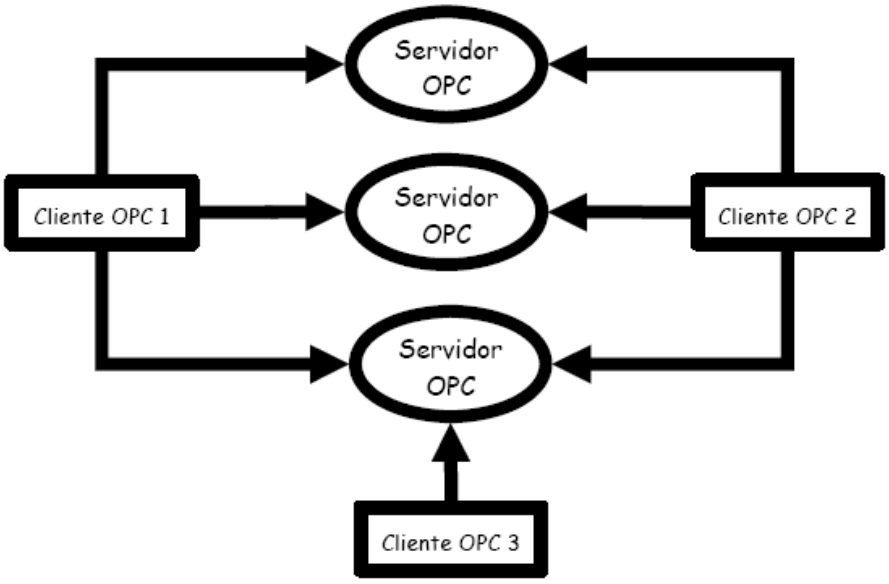


Figura 13 - Interacção entre aplicações Cliente e Servidor [Fonseca, 2002].

A especificação *OPC DA* define três atributos necessários para comunicação entre aplicações cliente e servidor [OPC, 2003]**:**

- Valor dos dados: todas as variáveis em formato *Integer*, *float*, etc;
- *Time Stamp*: informação fornecida pelo servidor, da leitura do *time stamp* dos dispositivos de campo ou por geração interna. É utilizada a estrutura normalizada do *Windows* para o *UTC* (*Universal Time Coordinated*);
- Informação de estado: indicação da qualidade dos dados fornecidos pelo servidor (Bom, Mau ou Desconhecido).

Configuração dos dados *OPC* no Cliente

De forma geral, os produtos de mercado não permitem muita flexibilidade para a configuração dos dados solicitados pelo cliente, o que se deve, sobretudo, à preservação dos modelos dos *drivers* de comunicação específicos. Considerando o caso mais comum, que consiste nos servidores de dados *OPC* (*OPC Data Access*), os clientes podem definir basicamente as seguintes configurações [DeltaV, 2007]:

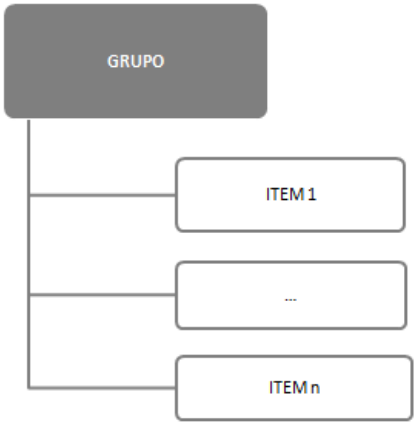


Figura 14 - Relação Grupo/Item (Tag).

- Criação de grupos e itens *OPC*, os diversos itens (*tags*) são organizados em grupos *OPC* (Figura 14), os quais definem as principais características de leitura dos itens;
- Leitura Síncrona ou Assíncrona;
- Permite a leitura de dados directamente ao dispositivo;
- Estado Activo/Inactivo: cada item ou grupo pode ter o seu estado alterado pelo cliente para activo, habilitando a comunicação do mesmo, ou inactivo;
- Leitura cíclica ou por mudança de estado;
- Definição de valores limites de transição para os itens de um determinado grupo, para os quais o servidor fará o envio para os clientes quando a alteração dos valores dos itens estiver fora da gama especificada;
- Escrita de dados *OPC* funciona de forma independente da leitura;
- Comunicação de blocos de dados reduzindo assim o *overhead* da comunicação. Neste caso, cada item é configurado como um bloco de dados.

2.3.2 *OPC XML Data Access*

A *OPC Foundation* já havia definido os interfaces para *Data Access Servers, Event Servers, Batch Servers, e History Data Access Servers*, servidores que contem informação valiosa para empresas e que está a ser fornecida a aplicações empresariais via interfaces baseadas na tecnologia *OLE/COM*.

A linguagem *XML* oferece um novo significado para descrever a troca estruturada de informação entre aplicações. *XML* é uma tecnologia facilmente integrada e disponível em diferentes plataformas.

A OPC XML Data Access (OPC XML-DA) é a adopção da *OPC Foundation* da tecnologia *XML* e das suas definições para facilitar a troca de dados dos dispositivos de campo, através da internet, para o domínio empresarial [OPC Training Institute, 2007].

Para aplicações que apenas carecem da funcionalidade de aceder aos dados (OPC DA), a especificação *OPC XML DA* é uma alternativa viável. Baseada em .NET3, a especificação *XML DA* tem a mesma segurança que a especificação *OPC UA, e os* aplicativos *DCOM* não são usados, eliminando os problemas das aplicações do tipo *DCOM*.

A especificação *OPC XML DA* permite criar de forma simples uma aplicação do tipo Cliente/Servidor, tendo sido modelada para que a informação estruturada seja entregue numa mensagem *SOAP* como um corpo (*Body*) *SOAP* [Advosol, 2008].

A referida especificação é análoga a uma especificação de automação (*Automation Specification*), que é um documento próximo a uma especificação *Custom* (*Custom Specification*). A especificação *Custom* disponibiliza os conceitos e capacidades base para a interacção entre cliente e servidor *OPC*. A especificação *OPC* vem então esclarecer como os conceitos e capacidades base são expostos como interfaces de automação, na especificação de automação, desta forma a *OPC XML – DA* é um complemento ao *OPC Data Access* 3.0.

O modelo da especificação *OPC XML DA* tem por base uma subscrição do tipo "*polled-pull*". O cliente estabelece um "contracto", uma ligação falsa com o servidor, isto porque a comunicação entre os dois não é permanente. O cliente inicia a subscrição e acorda a execução de pedidos periódicos para a actualização dos dados [OPC, 2004].

Para poder simular as capacidades de resposta do modelo inicial da *OPC DA* baseado na tecnologia *COM*, foi delineado um mecanismo de controlo das respostas para o modelo da *OPC XML DA*. O mecanismo pode ser usado para diminuir os tempos de resposta de alteração de dados para o cliente e minimizar o número de chamadas entre cliente e servidor.

Esta especificação suporta serviços baseados na subscrição: "*Subscribe", "SubscritionPolledRefresh"* e *"SubscriptionCancel".*

O serviço "*Subscribe*" é recurso para iniciar a ligação com o servidor. O serviço "*SubscritionPolledRefresh*" é usado periodicamente para receber as últimas alterações nos valores dos dados.Para terminar a ligação com o servidor usa-se o "*SubscriptionCancel*" .

A cada serviço segue-se um "*pull*" para obter o resultado, ou seja, a cada serviço executado segue-se um *GetResponse,* por exemplo a seguir a um *Read* segue-se um *ReadResponse* para obter o resultado.

Segue-se uma tabela com os serviços mais comuns para as interfaces baseadas na especificação *XML-DA* [OPC, 2004].

Tabela 2 - Descrição de serviços relativos à especificação OPC XML-DA.

Serviço	Descrição
Browse	Questiona algumas propriedades do servidor e todos.
GetProperties	Devolve informação extra associada a um ou mais itens.
GetStatus	Devolve informações sobre o servidor, versão, modo de operação, estado de funcionamento, etc.
Read	Devolve o valor para um ou mais itens.
Subscribe	Indica uma lista de itens para os quais o cliente pretende actualizações contínuas.
SubscriptionCancel	Remove uma lista de itens previamente definida pelo cliente com o serviço “Subscribe”.
SubscriptionPolledRefresh	Devolve todos os itens, indicados com o serviço “Subscribe”, que se alteraram desde o último “SubscriptionPolledRefresh”.
Write	Escreve novos valores para um ou mais itens.

A especificação OPX *XML-DA* é a primeira da *OPC Foundation* relativa aos *Web services*. Esta especificação está mais próxima da popular especificação *OPC Data Access*. A interface *Data Access* é usada para a leitura e escrita de dados em dispositivos típicos de automação, tais com *PLC's*, enquanto a *OPC XML DA* cria esta mesma interacção mas através da *Web*, ou seja, esta especificação apresenta-se optimizada para *Web services*.

2.3.3 *OPC* Unified Architecture

A especificação *OPC Unified Architecture* (*OPC UA*) é a nova norma definida pela *OPC Foundation*, para comunicação de dados em processos automatizados, diferindo significativamente dos seus antecessores. Após três anos de trabalho na especificação e outro ano de prototipagem, a nova especificação foi lançada.

É esperado que esta especificação substitua as bem sucedidas especificações baseadas na tecnologia *DCOM* da *Microsoft* da *OPC Foundation*, - a *OPC Data Access,* a *OPC Historical Data Access* e a *OPC Alarms and Events,* pois a *OPC UA* une todas as funções disponibilizadas pelas outras especificações [Advosol, 2008].

A nova especificação cria a base para uma inovadora plataforma de comunicação independente da tecnologia de informação. As possíveis aplicações nos níveis *MES* e *ERP* resultam em novos desafios em termos de interoperabilidade, capacitação na rede, independência de plataforma e segurança. Com estas características, espera-se que a *OPC UA* seja utilizada numa gama mais vasta da indústria e aplicações, do que as especificações clássicas [Mahnke, 2009].

A *OPC* UA permite assim maior interoperabilidade e unifica o uso de diferentes servidores *OPC* e clientes.

O *OPC UA* persegue dois objectivos:

1. Melhorar a compatibilidade e a combinação das especificações *OPC* existentes;
2. Alargar as possibilidades e benefícios das especificações *OPC* individuais.

Melhorar as possibilidades de combinar as diferentes categorias *OPC* server é o desejado, visto que os três servidores (*DA, AE* e *HD)A* são usados de forma crescente e simultânea em projectos, nos dias que correm.

A *OPC Foundation* decidiu definir um modelo que combina várias faces das especificações existentes para especializar um conjunto de serviços que são executados num único modelo de dados coerente.

Estes serviços oferecem capacidades muito semelhantes às interfaces existentes, *OPC DA, AE e HDA*, mas de uma maneira unificada. O modelo de dados único visa permitir combinar toda a hierarquia de *namespaces* para uma informação completa. Em adição, é definido um mecanismo uniforme para encontrar e ligar todas as fontes de dados [Lange, 2006].

A *OPC* UA também contém a tecnologia *Web Service*, incluindo algumas das normas mais recentes do *Web services*, que permitem que clientes e servidores *OPC UA* negociem, em processamento, quais os protocolos e a codificação suportada. A definição assegura a melhor comunicação possível, enquanto permite também uma maior interoperabilidade [Mahnke, 2009].

A especificação *UA* também usa eventos e *Web services* para suportar funções do tipo *callback*, como se usa nas interfaces do tipo *COM*, e como o *Polled refresh* definido na especificação *XML*-DA [OPC, 2008].

A *OPC UA* define dois tipos de comunicação possíveis, através de *SOAP* com os *Web services* sobre o *HTTP*, e para um desempenho significativamente melhor, a comunicação pode ser feita via *TCP*.

Para eliminar desvantagens da performance das comunicações baseadas nos *Web services* em comparação com *DCOM*, a *OPC UA* definiu o protocolo *UA* binário, que confere às aplicações *OPC UA* a capacidade de atingir performances semelhantes às *DCOM*, mas também suporta a comunicação sobre o protocolo *XML* comum [Lange, 2006].

A especificação *OPC UA* está definida em 12 partes, em que as primeiras sete (7) especificam as capacidades do núcleo da *OPC UA*. As referidas capacidades estabelecem a estrutura do *OPC AddressSpace* e dos serviços que nele operam. Da parte oito (8) à onze (11), as capacidades definidas são aplicadas aos tipos específicos de acesso delineados separadamente pelas especificações *OPC COM*, tais como a *DA, AE* e *HDA*. A parte doze (12) descreve os mecanismos de descoberta para a *OPC UA* [OPC, 2008].

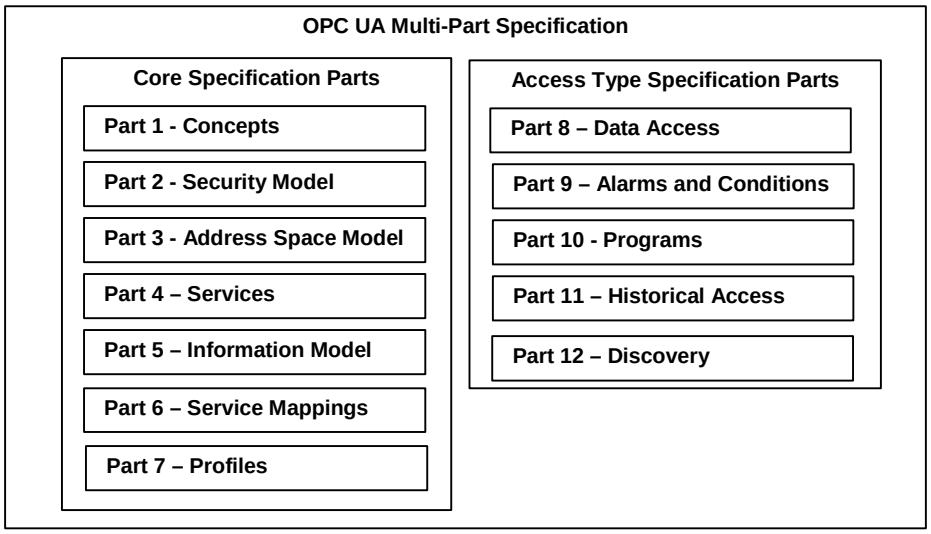


Figura 15 - Organização da especificação *OPC UA* [OPC, 2008].

Especificação do núcleo [OPC, 2008]

- Parte 1 – *Overview and Concepts* – Apresenta os conceitos e uma descrição da *OPC UA*;
- Parte 2 – *Security Model* – Descreve o modelo para segurança de interacções entre *OPC UA* Clientes e Servidores;
- Parte 3 – *Address Space Model* – Descreve os conteúdos e a estrutura da *AddressSpace* dos servidores;
- Parte 4 – *Services* – Específica os serviços disponibilizados por servidores *OPC UA*;
- Parte 5 – *Information Model* – Específica os tipos e as relações definidas para servidores *OPC UA*;
- Parte 6 – *Mappings* – Específica os mapas de transporte e a codificação de dados suportada pela *OPC UA*;
- Parte 7 – *Profiles* – Específica os perfis que estão disponíveis para clientes e servidores *OPC*. Esses perfis fornecem grupos de serviços ou funcionalidades que podem ser usadas em conformidade com o nível de certificação. Servidores e clientes serão testados contra os perfis.

Especificação do tipo de Acesso [OPC, 2008]

- Parte 8 – *Data Access* – Específica o uso da *OPC UA* para acesso a dados;
- Parte 9 – *Alarms and Conditions* - Específica o uso da *OPC UA* para aceder a alarmes e condições. O sistema base inclui suporte para eventos simples, alarmes e condições;
- Parte 10 – *Programs* – Específica o suporte da *OPC UA* para aceder a programas;
- Parte 11 – *Historical Access* – Determina a utilização da *OPC UA* para aceder a históricos, o que inclui os históricos de dados e de eventos;

- Parte 12 – *Discovery* – Define como operam os servidores de procura em diferentes cenários e descreve como Clientes e Servidores *OPC UA* devem interagir com eles. Também define como a informação relacionada deve ser consultada, usando protocolos como o *UDDI*.

Em suma, a *OPC UA* é aplicável a *software*s de produção em áreas como os dispositivos de campo, sistemas de controlo, *MES e ERP*. Os sistemas são definidos para trocar informação e para possibilitar o controlo e a supervisão para processos industriais.

A especificação *OPC UA* define um modelo comum de infra-estrutura para facilitar a troca de informação, definindo [OPC, 2008]:

- O modelo de informação para representar a estrutura, comportamento e semântica;
- O modelo de mensagem para interagir entre aplicações;
- O modelo de comunicação para transferir os dados entre os terminais do sistema;
- O modelo de conformidade para garantir a interoperabilidade entre sistemas;

O sucesso da *OPC UA* vai depender substancialmente da forma como os novos investimentos nos produtos *OPC*, que estão disponíveis hoje em dia, serão protegidos no futuro.

Desde o início que a *OPC Foundation* persegue uma estratégia de migração na forma de *Wrappers UA*. Um *Wrapper UA* é como que uma "casca" em volta dos produtos *OPC DCOM* existentes, permitindo a comunicação com os futuros produtos *OPC UA*. Num futuro próximo, os *Wrappers UA* permitirão que clientes *OPC DCOM* acedam a servidores *OPC UA* e que clientes *OPC UA* acedam a servidores *DCOM*, sendo que os produtos existentes não terão que sofrer modificações.

Desta forma, os produtos *OPC UA* poderão não só coexistir com os produtos *OPC DCOM*, mas também melhorar as capacidades dos mesmos, o que coloca a especificação *OPC UA* no melhor caminho possível.

2.3.4 Cenários de aplicação dos sistemas *OPC*

Actualmente soluções baseadas na tecnologia *OPC* são muito comuns na indústria desde habitações, centrais energéticas, etc, como também noutras áreas.

Através de ferramentas específicas para cada tipo de aplicação, os sistemas *OPC* aderem perfeitamente aos mais exigentes padrões de aplicações de controlo e supervisão de vários tipos de indústrias e segmentos.

Segue-se uma breve descrição das áreas onde é possível encontrar sistemas baseados na tecnologia *OPC*:

- Sistemas eléctricos e de energia;

- Farmacêutica e alimentar;
- Química e petroquímica;
- Saneamento;
- Produção;
- Prédios Inteligentes.

Alguns produtos *OPC* existentes no mercado

Kepware [Kepware, 2008b]

O *Kepware* é um dos *software*s, baseados nas especificações *OPC*, mais usados em todo o mundo para comunicação para automação. Desde o seu lançamento, em 1995, que a *Kepware* se focou no desenvolvimento de drivers de comunicação para os mais variados *PLC´*s e dispositivos de E/S. Para além disso, é compatível com grande parte dos sistemas operativos e dispõe de mais de 130 protocolos de comunicação.

A arquitectura, única, de driver combinada com as normas das marcas garante que o *Kepware* possa ser considerado uma solução ideal para as mais variadas aplicações. Os *software*s *OPC* da *Kepware* são conhecidos pela fiabilidade, qualidade e facilidade de uso.
O *Kepware* é caracterizado por uma interface única que torna concretizável maximizar os modelos e objectivos *OPC*. É possível criar vários drivers, todos no mesmo servidor *OPC*, sem necessidade de conhecer os protocolos dos diferentes dispositivos ou perceber novas aplicações.

A lista de drivers inclui *PLC's* e outros dispositivos, bases de dados e aplicação de drives específicas e a comunicação com os dispositivos e drive própria para definir a troca de dados para comunicação série ou *Ethernet* pelo utilizador.

Características:

- Baseado nas especificações OPC;
- Testado com inúmeras aplicações cliente;
- Definição avançada de Tag's;
- Janela de diagnóstico avançado de protocolos;
- Alterações on-line;
- Configurações podem ser protegidas por password;
- Possibilidade de criar uma rápida aplicação cliente para teste do programa;
- Design multi-tarefas permite uma óptima performance;
- Cada driver suporta até 16 comunicações em simultâneo;

- Drivers para dispositivos de enumeras marcas, tais como: Telemecanique, Siemens, Modicon, Honeywell, Toshiba, Mitsubishi, OMRON, Allen Bradley, *Ethernet* I/O, Texas Instruments, etc.

Mx OPC server [Mitsubishi, 2003]

O *Mx OPC server* contém todos os protocolos disponíveis para comunicar com sucesso entre uma qualquer aplicação cliente e um dispositivo da *Mitsubishi*. Através do *Mx OPC server* torna-se prescindível todo o desenvolvimento que era necessário para efectuar uma ligação entre uma aplicação e um dispositivo e criação do protocolo dedicado.

O *Mx OPC server* baseia-se nas especificações *OPC* e permite a transparência de dados com os mais variados *software*s baseados na tecnologia *OPC* e outros produtos tais como bases de dados e algumas ferramentas do *Microsoft Office*.

O *MX* é capaz de manter várias ligações ou mesmo uma rede com vários clientes, trocando dados entre eles. As ligações aos dispositivos podem ser feitas através das portas de comunicação série, *Ethernet, MELSECNET/10, CC-Link* entre outros.

O Mx OPC server foi desenvolvido para ser o servidor OPC Definitivo para os PLC's da Mitsubishi, tendo sido desenhado especificamente para ser intuitivo e fácil de usar, independentemente do dispositivo ou aplicação cliente. Para além disto, está bem preparado para integrar aplicações desenvolvidas em C, C++, Visual Basic, e Visual C++.

A estrutura de um projecto no *Mx OPC server* é muito simples e fundamentada na especificação *OPC DA.*

Características:

- Com base nas especificações *OPC;*
  - *OPC Data Access;*
  - *OPC Alarm & Events;*
  - *OPC XML DA.*
- Criação automática de blocos de dados;
- Capacidade de configuração e controlo local;
- Possibilita a comunicação via modem;
- Preparado para a leitura de dados a partir de dispositivos Modbus Ethernet e série;
- Pode activar ou desactivar dispositivos, blocos de dados e tags, individualmente;
- Usa o conceito de tags dinâmicas, ou seja, as tags não necessitam de ser criadas no configurador e são sempre associadas ao dispositivo, sendo apenas criadas e chamadas quando a aplicação relativa ao dispositivo está activa, evitando assim sobrecarregar a base de dados;
- Permite a leitura de todo o tipo de variáveis;

- Suporta diagnósticos avançados;
- Fornece informação em tempo real sobre estado dos dispositivos e ligações;
- O endereço dos dispositivos é detectado automaticamente;
- Leitura e escrita directamente nos dispositivos;

CX-server OPC [Omron, 2009]*,* [OMRON, 2008]

O *CX-server OPC* é um servidor *OPC* da *Omron* que recorre às normas de comunicação *OPC* e permite uma comunicação mais fácil e intuitiva com os sistemas de controlo *Omron*.

O *CX-server* possibilita desenhar, criar e manter sistemas de controlo e aquisição de dados, através de redes e fundamenta-se nas especificações *OPC*.

O *CX-server* apresenta a grande vantagem de ser possível usar *PLC's Omron* e outros dispositivos *Omron* numa arquitectura de rede aberta. Esta arquitectura é baseada nas normas *OPC* e foi desenhada para tornar viável a integração de outros sistemas na rede.

O *CX-server* disponibiliza os meios para efectuar as ligações numa rede aberta, e conferir a troca de dados entre *PLC's* e outros dispositivos, sendo ainda possível a troca de dados com outros servidores *OPC*.

Com o servidor *CX-server* pode-se definir e criar sistemas que controlam e monitorizam desde dispositivos, máquinas, a linhas inteiras de produção, permitindo a troca de dados em tempo real e estar ligado entre dispositivos de campo e os sistemas administrativos.

O sistema *OPC* da *Omron* não é um sistema rígido, pode ser facilmente adaptado a mudanças que possam surgir. Pode ser definida e implementada qualquer adaptação ou extensão para permitir o controlo e supervisão de todo o tipo de dados.

Principais características:

- Disponibilizar as informações de produção em software normal de escritório;
- Acesso fácil a todas as variáveis de PLC's e, em consequência, a todas as informações das máquinas, produção e processos;
- Verificar e alterar o estado dos dispositivos;
- Verificação do estado das ligações;
- Conhecimentos de programação ao nível do software da Microsoft, com o objectivo de reduzir o tempo de formação e evitar o recurso a especialistas;
- Automatizar a criação de relatórios de produção morosos;
- Criar uma aplicação de visualização HMI de uma só máquina com software normal da Microsoft e utilizar o OPC para ligação com o sistema de controlo;
- Controlo de Supervisão e Acesso a Dados;

INGEAR OPC Server [Ingear, 2006]

As soluções *Ingear OPC Server* permitem a ligação a *PLC's* da *Allen-Bradley, Modbus, GE*, e *Omron* aos sistemas das empresas. Com as especificações *OPC* é possível ligar os *PLC's* a aplicações *HMI*/*SCADA*, aplicações cliente de outras marcas e também a aplicações cliente *OPC* personalizadas.

O *Ingear* é baseado na tecnologia *OPC* e garante interoperabilidade entre sistemas, fornece uma plataforma cruzada sobre as comunicações série e *Ethernet* para grande parte dos sistemas operativos, quer para computadores quer para dispositivos móveis.

Em todas as versões do *Ingear* não há necessidade de outros *software*s ou drives para que seja possível comunicar com os *PLC´*s, pois o *Ingear* dispõe dos seus próprios drives de comunicação com os dispositivos.

O *Ingear* permite assim a conectividade com a maioria dos sistemas operativos e permite a comunicação com praticamente todos os *PLC's* das seguintes marcas:

- *Allen-Bradley ;*
- *Modicon PLCs & Modbus Devices;*
- *Omron;*
- *GE Fanuc.*

Características comuns:

- Baseado nas especificações *OPC*, especialmente na *OPC Data Access*;
- Interoperabilidade *OPC* entre sistemas;
- Endereçamento de *Tag* (Variáveis);
- Importação e exportação de *Tags*;
- Suporta modo de simulação;
- Compatível com vários sistemas operativos;
- Configuração para sistemas operativos móveis (*PocketPC*), igual aos restantes sistemas operativos, sendo possível importar as configurações;
- Extremamente intuitivo e de fácil uso;
- Funções de monitorização de dados e simulação reduzem o tempo de desenvolvimento;
- Um único servidor suporta várias interfaces de comunicação;
- Não precisa de outros softwares ou drives;
- Optimizado para operações de leitura e escrita de blocos;
- Suporta vários clientes *OPC.*

Elipse E3 [Maciel, 2003], [Elipse, 2008]

O Elipse E3 é um servidor completo de supervisão e controlo de processos, com muitas possibilidades de ligação, muito flexível e fiável, sendo um *software* que representa o resultado de anos de trabalho e pesquisa da Elipse *Software*. A sua arquitectura distribuída, com funcionamento em rede, totalmente transparente, compõe um verdadeiro sistema multicamadas, oferecendo uma plataforma de rápido desenvolvimento de aplicações, alta capacidade de comunicação e garantia de expansão, preservando assim os investimentos das empresas.

Principais vantagens

- O *Elipse E3* permite a comunicação com inúmeros protocolos e equipamentos, para além de que pode englobar sistemas locais ou mesmo remotos;
- As aplicações desenvolvidas com o *Elipse E3* utilizam orientação a objectos, o que aumenta a qualidade da aplicação, reduzindo os custos de programação e manutenção dos sistemas;
- Com o *E3* é possível construir aplicações com enorme capacidade de actualização e evolução constante, sejam elas simples *HMI* ou complexos sistemas de operações em tempo-real;
- Arquitectura exclusiva Cliente-Servidor com rede transparente, sem necessidade de copiar os interfaces entre as estações.

***Software HMI/SCADA* para aplicações avançadas e distribuídas**

- Servidores robustos, que recolhem, processam e distribuem dados em tempo real;
- Interface de operação independente através do *E3 Viewer, Internet Explorer* ou serviços de terminal;
- Arquitectura distribuída;
- 100% *Internet-ready*.

Arquitectura totalmente orientada a objectos

- Uso extensivo de bibliotecas do utilizador, com criação de galerias e modelos de objectos gráficos, bem como e estruturas de dados com capacidade de inclusão de qualquer funcionalidade;
- Mais de 3000 símbolos gráficos;
- Configuração on-line;
- Bases de dados abertas: o *E3* não utiliza formatos proprietários;
- Poderosa ferramenta de relatórios incluída;
- Extensa biblioteca de símbolos;

- Completa gestão de alarmes;
- *OPC (OLE for Process Control*) Cliente e Servidor;
- Criação de Históricos dos processos;
- Módulo de elaboração de relatórios;
- Suporte para componentes *ActiveX*.

2.4 Outros trabalhos

2.4.1 Trabalhos da autoria e co-autoria do autor da presente tese

Neste capítulo abordaremos alguns trabalhos já desenvolvidos pelo autor da tese em anos anteriores e que podem fundamentar e ajudar na concepção deste trabalho.

Software SCADA [Costa, 2007]

O projecto *Software* SCADA consistiu no desenvolvimento de um *software*, em *Visual Basic*, de supervisão de ambientes industriais equipados com *PLC's* e foi criado no âmbito da disciplina de Projecto do 5º ano da Licenciatura em Engenharia Mecânica.

A razão que levou à elaboração do projecto "*Software* de Supervisão e Controlo de Redes Industriais" foi a necessidade de desenvolver um *software* genérico de supervisão e aquisição de dados. O *software* deve ser acessível às pequenas e médias empresas e permitir, de forma simples, a supervisão e controlo dos sistemas existentes na indústria, mas não apenas..

O projecto passou pelo desenvolvimento de rotinas de comunicação entre o *software* e diferentes tipos de *PLC's*: *Siemens* S7-200 e S7-300, *Mitsubishi*, *Omron*.

O *software* é caracterizado por:

- Interface gráfico simples e intuitivo, possibilitando a criação de páginas de supervisão definidas pelo utilizador do sistema. A interface é orientada por "objectos", possibilitando a sua selecção em função do tipo de variável que se pretende supervisionar:

- Variáveis digitais, visualização do estado de funcionamento dos equipamentos (automático, em ciclo, com ou sem defeito, etc.);
- Variáveis analógicas - permitir visualizar o valor em tempo real, e/ou realizar gráficos do seu valor;
- Histórico de defeitos e mensagens do sistema.

As rotinas de comunicação comunicam directamente com o CPU (Unidade de Processamento Central) dos *PLC's* sem que seja necessário realizar programação adicional nos *PLC* ou adicionar *hardware*.

Para que o grande objectivo de criar um *software* de supervisão e controlo fosse atingido, inicialmente realizaram-se programas de teste, para cada marca de *PLC's*, recorrendo ao uso de *OPC* Servers, *Kepware* para *Siemens*, *Mx OPC server* da *Mitsubishi* e *Ingear* para o *Omron*. Estes programas permitiam efectuar a supervisão e controlo dos *PLC´s* e desta forma obter uma maior familiarização com os dispositivos em causa e obter maior compreensão de uma estrutura para sistemas de supervisão e controlo.

Seguiu-se a implementação das rotinas para comunicação directa com os CPU's de cada autómato e a implementação de pequenos programas de teste que permitiam a leitura e escrita de todo o tipo de variáveis. Para permitir testar os programas foi reestruturada uma bancada de ensaios, disponibilizada pela empresa Atena. Tendo a bancada funcionar plenamente, com quatro *PLC's* e as rotinas de comunicação desenvolvidas, de forma independente, avançou-se para a elaboração do *software*. O desenvolvimento do *software* impôs a criação de uma estrutura de programação que permitisse o uso de uma grande quantidade de variáveis e fosse possível adaptar de forma simples às necessidades de um *software* de supervisão e controlo. Por outro lado, a estrutura tinha de facilitar e permitir a interoperabilidade entre vários sistemas.

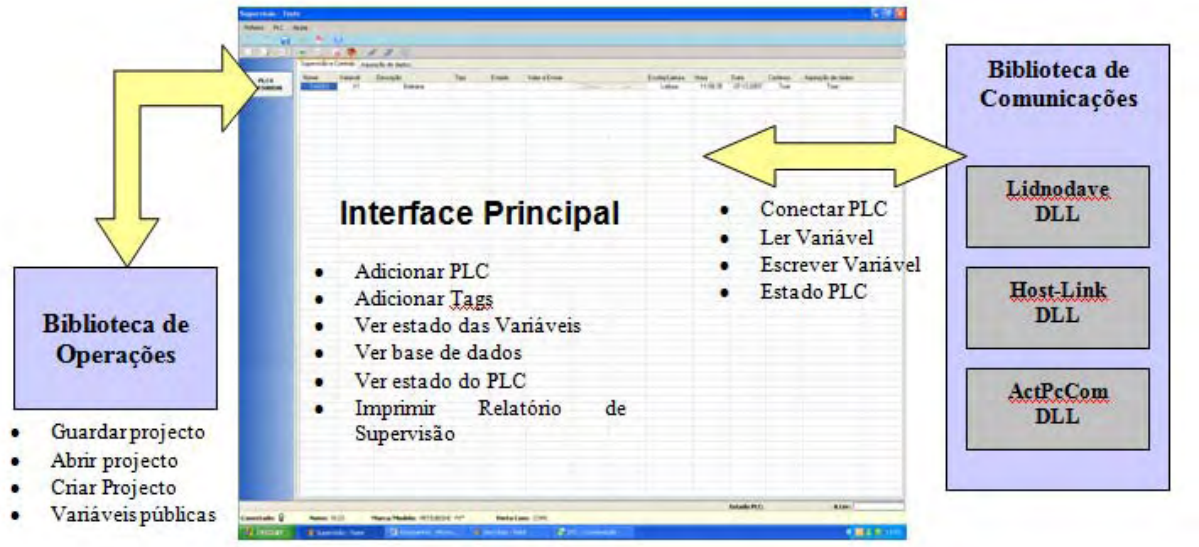


Figura 16 - Estrutura e interface do software de supervisão [Costa, 2007].

A comunicação com os vários *PLC's* era feita através de portas série virtuais, tendo em conta que o modo de comunicação com os vários *PLC's* era por meio da porta série. Todavia, para tornar possível efectuar a ligação a mais que um *PLC* e sem necessidade de estar junto do *PLC*, optou-se por utilizar um dispositivo de conversão *Ethernet*-Série, ao qual era possível ligar os quatro *PLC's* e, através da rede, adquirir IP e criar portas série virtuais, para assim viabilizar a comunicação com os *PLC's*.

Com o intuito de facilitar o teste do *software* e efectuar melhorias foi restaurada e remodelada uma bancada de testes que tinha implementado um ciclo contínuo de medição e comparação de peças circulares e quadradas. A bancada de testes era constituída por uma estrutura em perfil de alumínio, com dois quadros eléctricos e um quadro com a instalação pneumática. Nos quadros eléctricos destacam-se as duas fontes de tensão, o módulo de segurança *Pilz Pnoz X3*, o variador de velocidades da *Telemecanique* e quatro autómatos, dois *Siemens (S7-314 e S7-224)*, um *Omron CPM2A* e um *Mitsubishi FX2N*.

Na base da bancada encontrava-se o sistema de visão *Omron* e todos os cilindros pneumáticos e o tapete accionado por um motor DC que executam o ciclo contínuo de medição. Para além destes componentes existiam duas estações de medida dimensional *Etamic Digitamic* para efectuar as medições, bem como o painel de controlo onde estão colocados botões de pressão e duas consolas para supervisão e controlo da bancada.

No final da implementação do *software* e após vários testes, foi possível considerar que o *software* apresenta uma interface simples, intuitiva e funcional, preparada para a supervisão e controlo de qualquer sistema com *PLC's*, dentro das gamas e marcas mencionadas.

Protocolos de Comunicação CN [Costa, 2007b]

No ano de 2006 no âmbito da disciplina de Comando Numérico (CN) computorizado, foi realizado um trabalho sobre protocolos de comunicação com comandos numéricos *Siemens*, *Fagor, Fanuc e Heidenhein*. Para além do levantamento dos protocolos foi ainda elaborado um programa que permitia a comunicação com um comando numérico Fagor, através do protocolo DNC50.

O programa conferia a leitura e a escrita de variáveis do comando numérico e do respectivo *PLC*, para além de que era possível ler programas ISO existentes no CN, transferir programas para o CN e executar os programas pretendidos.

A ligação com o CN era feita utilizando a porta série, uma vez que o modelo do CN apenas dispunha de uma porta série. Embora seja um tipo de comunicação com algumas limitações, é ainda muito útil e de fácil integração.

Este programa permitia a transferência de programas ISO para o CN de forma simples, sendo que era suficiente ter o programa ISO num documento de texto com a extensão ".txt" e abri-lo no programa, ou escrever o programa ISO na área de trabalho do programa e de seguida enviá-lo para o CN, evitando assim criar os programas ISO no CN ou recorrer a outros meios ou *Software* da Fagor.

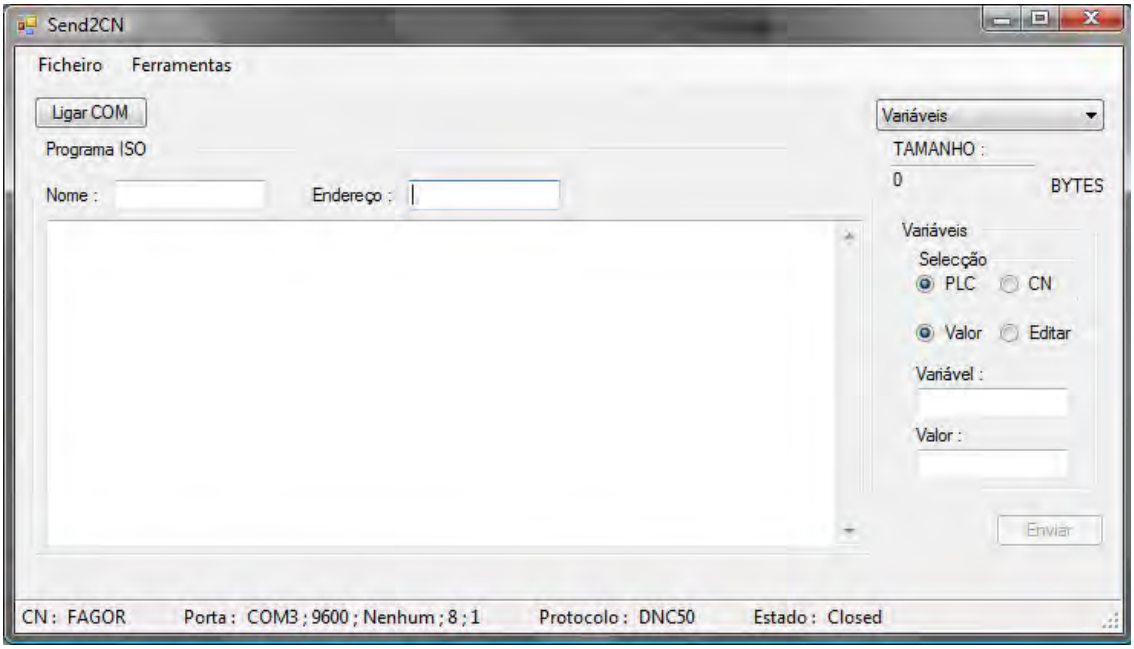


Figura 17 - Janela principal do programa de comunicação com CN's da Fagor.

Para efectuar leitura e escrita de variáveis o programa dispunha de um campo exclusivo onde era inserida a variável a ler ou escrever e respectivo valor.

Nos testes, o programa mostrou-se bastante eficaz para o objectivo pretendido, de fácil de uso e intuitivo. As suas rotinas de comunicação também foram desenvolvidas de maneira a serem facilmente adaptadas para outras soluções.

Supervisão e controlo de dispositivos via *Web* com recurso a servidores *OPC*

[Costa, 2006]

No âmbito da disciplina de Informática Industrial foram desenvolvidas duas aplicações para permitir a supervisão e controlo de dois *PLC's*, um *Siemens S7 314* e um *Omron CPM2A* através de uma página *Web*, programada em *PHP* e *HTML*.

As aplicações foram desenvolvidas em Visual Basic.Net, sendo que a comunicação com os *PLC's* era feita através dos servidores *OPC Kepware* para o *Siemens S7 314* e o servidor *OPC Ingear* para o *Omron CPM2A*.

Para cada *PLC* foi criado um programa de forma a simular um pequeno recurso (linha simples de enchimento de garrafas no *Siemens* e controlo de um motor no *Omron*). No passo seguinte foi possível verificar o estado desse recurso e controlá-lo via *Web* e localmente, com a leitura e escrita de variáveis nos *PLC's* através dos servidores *OPC*.

A interacção entre as aplicações locais e a aplicação *Web* (página *HTML*) foi realizada com recurso à base de dados do *MySQL*, tendo sido criada no computador que estava ligado ao *PLC* através da porta série.

A base de dados tinha tabelas para a troca de informação entre as aplicações, que permitiam a supervisão contínua e o controlo do recurso via *Web*.

A implementação deste sistema foi bem sucedida e serviu para uma melhor compreensão dos servidores *OPC* e verificar que a possibilidade de supervisão e controlo de um recurso através de uma página *Web* é bem possível e praticamente em tempo real, existindo apenas algum *delay (atraso)*, relativo à troca de informações e tempos de escrita e leitura das tabelas, da base de dados, que servem de meio de troca de informação entre aplicação local e cliente *Web*.

2.4.2 Trabalhos ou teses desenvolvidos por outros autores ou entidades

Neste capítulo, e como o nome indica serão apresentados trabalhos ou teses desenvolvidos na área dos serviços *Web* ou *software*s *SCADA* ou tecnologia *OPC*.

Survey of SCADA SYSTEMS and visualization of a real life process

[Gomez, 2002]

O trabalho de investigação tinha como objectivo verificar a viabilidade e implementar um completo sistema *SCADA* numa fábrica de produção de carros de brincar da Lego.

No desenvolvimento do projecto, o autor verificou que o avanço da tecnologia disponível e o consequente avanço das linhas de produção e automatização das mesmas, levaram à existência de inúmeros pontos de aquisição de dados e vários sistemas e dispositivos de controlo de processos e recursos na unidade fabril em questão.

A existência de vários controladores (*RTU's*), na maioria *PLC's*, implicou que o autor fizesse um levantamento das soluções de *software* e ferramentas existentes para os *PLC's* mais comuns na unidade fabril, sendo na maioria *Siemens*, *Mitsubishi* e *Omron*.

De forma a desenvolver um sistema viável, o autor necessitou de analisar várias soluções que permitissem a comunicação com os diferentes *PLC's* e normalizar as comunicações com os *PLC's*, para permitir a intercomunicação entre sistemas.

A realização da monitorização da linha de produção, implicou que fossem analisados dois *software*s *SCADA* - o *WinCC* e o *Wonderware InTouch*, tendo optado pelo *InTounch*, apesar *WinCC* ser um *software* com mais potencialidades para possibilitar a supervisão de toda a linha, o *InTouch* é mais simples e pode integrar-se facilmente com outras aplicações.

No entanto, para possibilitar a flexibilidade do sistema de supervisão, e assim, permitir a interoperabilidade do sistema com todos os dispositivos existentes, o autor teve que recorrer a outra ferramenta, que permitisse após estabelecer as ligações físicas entre PC e *PLC's*, comunicar com os diferentes *PLC's*, uma vez que cada um tem o seu protocolo proprietário. A solução passou por utilizar o servidor *OPC Kepware* server. O *Kepware* permitiu a comunicação com os diferentes *PLC's* para efectuar a aquisição de dados e eliminou os problemas para estabelecer as comunicações.

Através do *InTouch* o autor conseguiu realizar um projecto de supervisão para todos os processos da linha de montagem de carros, sendo que o maior desafio com que o autor se deparou foi a solução para permitir comunicar com os diferentes *PLC's* e integrá-la no mesmo projecto *SCADA*.

Estudo de arquitecturas e desenvolvimento de ferramentas em plataformas *Mobile* para sistemas *SCADA* [Campos, 2006]

O referido trabalho teve como pressuposto criar um sistema *SCADA* para dispositivos móveis, ou seja, elaborar um sistema que possibilitasse, através de um telemóvel ou *PDA*, controlar e monitorizar um recurso ou um ou mais *RTU's*, principalmente em estações eléctricas e transporte de electricidade.

A crescente miniaturização da electrónica a nível mobile (*PDA's* e telemóveis) torna disponível, o desenvolvimento de novas ferramentas de engenharia e *HMI* para sistemas *SCADA*. Como é uma área com tecnologias recentes, ainda em implementação no mercado (por exemplo o *UMTS*), torna estas ferramentas mais inovadoras no mercado de produtos *SCADA*.

A utilização de telemóveis em sistemas *SCADA* é pouco comum, no entanto são principalmente utilizados para recepção de *SMS*. Mas interacção entre *RTU's* e telemóveis, para visualização de sinópticos, é ainda uma tecnologia em desenvolvimento.

A utilização de *PDA's* em *SCADA*, começa a ser utilizado e visto na *Web*, e apresenta-se como uma ferramenta bastante útil.

Os principais objectivos desta tese foram:

- Estudar tecnologias mobile disponíveis;
- Analisar e desenhar arquitecturas de interacção com sistemas SCADA;
- Realizar protótipos na área de engenharia (Diagnostico de RTU's, configurador de RTU's);
- Realizar protótipos na área de HMI (para PDA);
- Realizar protótipos na área de protocolos de comunicações (para PDA);
- Realizar protótipos na área de mini – HMI (para telemóveis);
- Estudar protótipos de hardware. No caso de fazer sentido comercialmente, fazer protótipos necessários para comunicação Wireless com RTU´s.
- Esta tese necessitou de uma grande pesquisa e desenvolvimento nas seguintes tecnologias:

- Windows pockets PC (Sistema Operativo);
- Visual Basic.NET 2003 (plataforma de programação);
- XML;
- J2ME (programação Java para mobile edition);
- Bluetooth / IRDA / RS-232;
- OLE / COM / OPC;
- Serviços Web;
- GSM / *UMTS*.

Interacção com uma Arquitectura de Componentes com Independência de Dispositivo e Localização [Pereira, 2005]

Nesta tese o autor verificou que, actualmente, há um crescente número de utilizadores de informação empresarial que se encontram afastados do seu escritório, pelo que se torna uma necessidade suportar novas formas de acesso a essa informação.

O aparecimento e o desenvolvimento de novos equipamentos móveis, como os telefones móveis e *PDA's*, vieram contribuir para o suporte deste cenário de mobilidade.

Os sistemas *SCADA*/DMS (Supervisory Control and Data Acquisition / Distribution Management Systems) possuem cenários de utilização em que o uso de dispositivos móveis é vantajoso. A utilização destes dispositivos mais pequenos, mais baratos, mas com limitações como a capacidade de apresentação ou a largura de banda suportada, exige às empresas um esforço na geração de novas interfaces para estes equipamentos.

O objectivo desta tese focou-se na apresentação de uma arquitectura, que suporte a interacção com um sistema *SCADA*/DMS, utilizando dispositivos variados com independência de localização. Esta arquitectura facilita a geração de interfaces para novos tipos de equipamentos, bem como a sua manutenção.

Da pesquisa tecnológica realizada, resultou uma arquitectura que utiliza a *Web*, em que o *XML* é o elemento comum, conferindo-lhe capacidades que resultam das características do *XML*. A clara separação entre a camada de apresentação e a camada de lógica de negócio, e a facilidade de inter-operação são as duas mais significativas.

Os serviços *Web* são utilizados para permitir a integração com a arquitectura de componentes e devido ao enorme potencial dos serviços *Web* e facilidade de integração, foi possível criar a camada intermédia de comunicação

Mediante a arquitectura proposta, foi implementada uma aplicação que permitisse a sua avaliação e exploração. Os resultados foram satisfatórios, cumprindo-se os requisitos que culminam do enunciado da tese: a independência de localização e a independência de dispositivo de interacção.

As conclusões mostraram a potencialidade de é possível suportar a interacção sobre o sistema *SCADA*/DMS, com independência de localização e de dispositivo.

Web services **– Metodologias de desenvolvimento** [Lopes, 2004]

Os autores do "Web services – Metodologias de desenvolvimento" fizeram um levantamento de várias plataformas possíveis de desenvolvimento de *Web services* e verificaram a sua viabilidade em diferentes plataformas.

Os *Web services* uma tecnologia emergente, e com enorme potencialidade, apresentando-se como uma tecnologia a seguir no desenvolvimento de aplicações distribuídas.

Os autores verificaram e classificaram os *Web services* como aplicações modulares, acessíveis através de um URL, independentes das plataformas de desenvolvimento e que permitem a interacção entre aplicações sem a intervenção humana, ou seja, os *Web services* apresentam-se como uma boa solução para problemas de integração e comunicação entre aplicações.

Os autores sublinham que as características supra mencionadas devem-se, sobretudo, ao facto dos *Web services* serem baseados em protocolos *normalizados*, entre os quais se destacam o *XML*, *SOAP*, *WSDL* e *UDDI*.

Os autores deste trabalho realizaram vários *Web services* e várias aplicações cliente, todos eles desenvolvidos em diversas e distintas plataformas- *SOAP*-Lite para *Perl*, Nu*SOAP* para *PHP, o WASP Server* para *Java* e *.Net* da *Microsoft.*

O trabalho permitiu concluir que existem diferenças no desenvolvimento dos *Web services* entre as várias plataformas, tendo sido apresentadas todas as divergências encontradas, no entanto, os *Web services* mostraram-se funcionais e fiáveis independentemente da plataforma de desenvolvimento, o que reforçou a caracterização dos *Web services* como funções flexíveis, com enorme potencialidade e que permitem e facilitam as comunicações entre sistemas.

Serviços *Web* [Pereira, 2004]

Descrever a arquitectura dos serviços *Web* foi o pressuposto do trabalho, no qual o autor apresenta a definição de serviços *Web* como uma interface que disponibiliza mecanismos normalizados de inter-operação entre distintas aplicações informáticas, correndo em distintas plataformas.

Os serviços *Web* são uma tecnologia que contorna uma parte significativa dos problemas encontrados quando se pretende distribuir o processamento por diversas máquinas, principalmente quando se trata de plataformas distintas e ou de ferramentas diferentes. Numa plataforma cuja base é a Internet, os

serviços *Web* são a solução mais indicada para o processamento distribuído, não só pela sua interoperabilidade, mas também pela facilidade de implementação e de acesso.

No desenvolvimento deste trabalho o autor apresentou os protocolos em que se baseiam os serviços *Web*, *XML*, *SOAP*, *UDDI* e *WSDL*. A estrutura da arquitectura de um serviço *Web* e o seu desenvolvimento também foram apresentados, tendo inclusive sido mostradas várias plataformas de desenvolvimento dos *Web services* (serviços *Web*). No trabalho foi ainda realizada uma análise profunda do desenvolvimento dos serviços na plataforma *.Net* da *Microsoft*

A *Microsoft*, através da plataforma .Net, veio facilitar de forma significativa o desenvolvimento de aplicações para Internet, especialmente as vocacionadas para os serviços *Web*. Apesar das tecnologias utilizadas na implementação dos serviços *Web* não serem facilmente assimiláveis do ponto de vista de especificação, a sua utilização é praticamente transparente, prescindindo assim de um programador de qualquer conhecimento das tecnologias utilizadas pelos serviços *Web*.

Software SCADA como plataforma para a racionalização inteligente de energia eléctrica em automação predial [URZÊDA, 2006]

A proposta deste trabalho focou-se no desenvolvimento de sistema inteligente, que através da tecnologia de automação predial e com recurso à monitorização de variáveis, em conjunto com uma estratégia de controlo, que pudesse gerir uma economia considerável do consumo de energia eléctrica nos ambientes fechados com aparelhos de ar condicionado, mantendo-se a preocupação com o conforto térmico. De referir que geralmente as variáveis são negligenciadas por sistemas de controlo tradicionais.

Para avançar com os testes, o autor necessitou de criar um pequeno sistema de monitorização em cada aparelho de ar condicionado, sendo composto por um medidor de energia e um *PLC*. De forma a permitir a comunicação com o *software* utilizado para realizar a monitorização (*ActionView*) foi instalada uma rede EIA 485 entre os *PLC's* e outra entre os medidores de energia, criando assim um sistema *SCADA*. A rede associada ao *software* seleccionado permitiu a comunicação com os *PLC's* e com os medidores de energia.

O autor conseguiu atingir o objectivo proposto para o trabalho, ao implementar um sistema *SCADA* que lhe possibilitasse reduzir e racionalizar os consumos de energia associados a aparelhos de ar condicionado. Os resultados foram positivos, todavia, mostraram a necessidade de testar outros modelos de controlo.

Integração de Sistema de Produção [Quintã, 2008]

A finalidade da referida tese prendeu-se com a análise de arquitecturas orientadas a serviços (*SOA*), para a integração de sistemas de produção, apesar de ainda não existir uma norma definida para este tipo de serviços.

A arquitectura orientada a serviços e particularmente e as suas implementações de *Web services* são um tema de pesquisa actual na área da automação industrial. O fornecimento de ferramentas independentes da plataforma e linguagem de desenvolvimento permitem a descentralização de recursos e serviços.

O autor analisou as várias arquitecturas orientadas a serviços, implementações de *Web services* e as suas normas *SOAP*, *WSDL* e *UDDI* tendo em vista a utilização para sistemas de produção flexíveis e descentralizados.

O trabalho envolveu a realização de infra-estruturas informáticas para três casos de estudo diferentes, um tapete de transporte automatizado, um centro de maquinação *Heidenhain* e um sistema flexível composto por vários recursos dispostos num *layout* definido, para os quais poderia enviar ordens de produção organizadas para todo o sistema.

O desenvolvimento do trabalho levou o autor a levar a cabo uma pesquisa pormenorizada sobre *Web services* e as suas normas *SOAP*, *WSDL* e *UDDI, como* também sobre tecnologias de comunicação tais como a *OPC*.

Os resultados experimentais mostraram a validade e o potencial de aplicação da arquitectura proposta para ambientes industriais e académicos. Os resultados evidenciaram a possibilidade de utilizar arquitecturas descentralizadas e orientadas a serviços modulares que pode mesmo substituir as arquitecturas orientadas a objectos.

3 Soluções propostas

De acordo com o explicado nos objectivos para este trabalho, pretende-se desenvolver um conjunto de serviços *Web* que permitam a um programador, mesmo sem conhecer a linguagem dos recursos, criar uma pequena aplicação *SCADA* que possa, remotamente, controlar e monitorizar um recurso, ou mesmo criar uma estrutura para controlar e monitorizar um sistema flexível de produção, numa perspectiva de controlo integrado de produção.

O objectivo levanta inúmeros problemas, para os quais será necessária uma solução que permita a interoperabilidade entre sistemas e recursos e, para além disso, que sejam de fácil aplicação, intuitivos e úteis.

A primeira questão que se impõe é relativa à arquitectura da solução a apresentar, ou seja, como efectuar a troca de dados entre os recursos e os clientes remotos? Na existência de intermediários, qual a sua posição na estrutura e qual a disposição de todos os componentes da arquitectura a propor?

A utilização de bases de dados também deve ser analisada com clareza visto que a optimização e a fiabilidade do sistema pode variar de acordo com a forma como se utilizam as bases de dados.

Relativamente aos dispositivos para os quais será possível utilizar os serviços *Web* será necessário seleccionar alguns dos equipamentos mais comuns na indústria e em ambientes académicos. Também foi necessário determinar como se efectuará a comunicação com os equipamentos e quais as funções para efectuar a comunicação na aplicação local.

Outro ponto sobre o qual é importante reflectir é a questão dos serviços *Web* a criar, qual o tipo de informação que deve ser trocada e como é trocada entre clientes e aplicação local.

Os serviços criados deverão permitir o acesso de clientes que desconheçam os equipamentos e as variáveis associadas a um processo, bem como os parâmetros necessários para realizar determinado processo, para além de que deverão disponibilizar as ferramentas necessárias a clientes que se encontram familiarizados com os dispositivos e suas variáveis e com os processos em questão, de tal forma que se saiba quais os devidos parâmetros para realizar determinada tarefa.

De um modo geral, o pretendido é criar dois tipos de *Web services*, ou seja, serão criados serviços que comuns aos equipamentos e independentes da marca ou modelo, de forma a que o cliente com poucos dados consiga gerir um processo. Por outro lado, serão desenvolvidos serviços para clientes familiarizados com os recursos e os programas de controlo dos mesmos e que saibam quais as variáveis e parâmetros para realizar determinadas tarefas ou mesmo forçar estados.

3.1 Arquitectura

A arquitectura da solução a propor permitirá a implementação de um sistema do tipo Cliente-Servidor e deverá permitir que uma aplicação cliente, remota, controle e monitorize um recurso de forma simples, fiável, que diminua ou mesmo elimine a perda de dados. Pretende-se que a arquitectura a desenvolver possibilite a interoperabilidade entre vários sistemas, aplicações e dispositivos.

A arquitectura será constituída por várias entidades a ser seleccionadas de acordo com a sua relevância e funcionalidade para o bom funcionamento de todo o sistema.

A arquitectura que se pretende será uma arquitectura orientada a serviços *SOA (Service Oriented Aplication)*, que funcionará como uma estrutura de suporte e apoio à integração de sistemas distribuídos e descentralizados.

Nos extremos opostos da arquitectura encontraremos um ou mais recursos e/ou a(s) aplicação(ões) cliente. Nas camadas intermédias existirão aplicações locais, módulos de comunicação ou rotinas para efectuar as comunicações com os recursos, *Web services* e Bases de Dados. A aplicação local será um *Middleware* que permitirá a troca de informação entre aplicação cliente e recurso/equipamento.

Todos estes componentes serão úteis ou mesmo obrigatórios para estabelecer a arquitectura que melhor se enquadre com as necessidades da solução a propor e com maior eficiência.

Os *Web services* e a aplicação local são componentes vitais de uma qualquer arquitectura, visto que os *Web services* são as ferramentas/funções que permitem a uma aplicação cliente, remota ou local, possibilitando efectuar pedidos, bem como receber as informações/respostas aos pedidos. A aplicação local é também essencial, tendo em conta que esta contém todos os módulos de comunicação e ligação para com os recursos. A aplicação local poderá também facultar a gestão de um ou mais recursos em função dos pedidos efectuados por aplicações cliente.

Apresentam-se várias hipóteses para a arquitectura do sistema a realizar.

Hipótese 1

A hipótese que se coloca em primeira-mão será uma arquitectura composta apenas por três componentes: o equipamento, os *Web services* disponíveis num servidor local e a aplicação cliente. A aplicação cliente utiliza os *Web services* como se fossem funções comuns e ao executar um serviço, se todos os parâmetros estiverem correctos, este comunicará com o recurso para cumprir a ordem pretendida. A comunicação com o recurso será feita por meio de um servidor local que disponibiliza os *Web services* e mantém a ligação física ao recurso, já os dados dos pedidos a realizar são transferidos da aplicação cliente para o servidor local via HTTP (ver Figura seguinte).

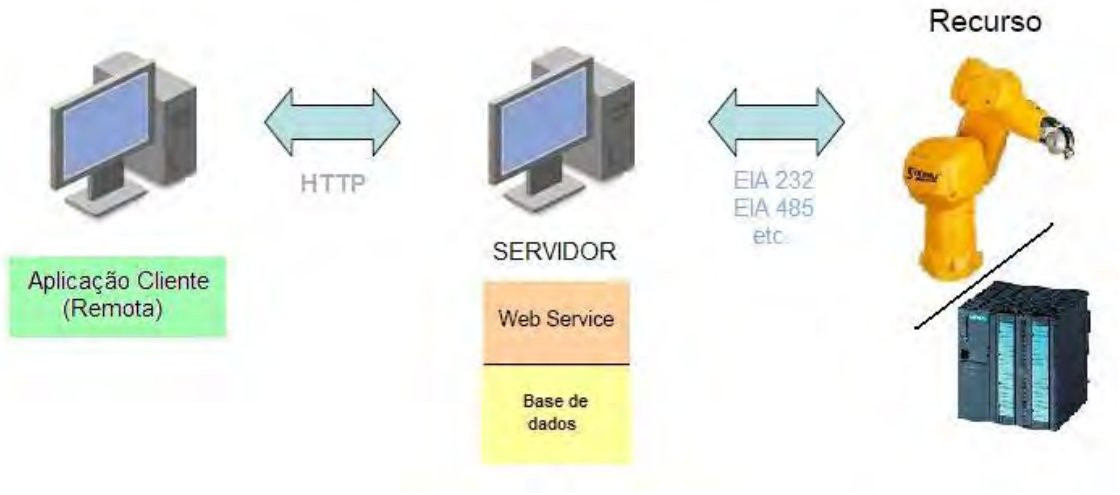


Figura 18 - Esquema da arquitectura proposta na hipótese 1.

Após comunicar com o *PLC* e obter uma resposta, o *Web service* armazenará as informações numa base de dados, para que a aplicação cliente execute um "*pull*" do resultado, tomando conhecimento dos dados relativos ao pedido efectuado ou o serviço pode devolver o resultado do pedido, não sendo necessário executar o "*pull*"dos dados.

O armazenamento das informações na base de dados é efectuado pelo serviço *Web* que define a ordem/tarefa a executar, sendo efectuado após a comunicação de forma a permitir obter um resultado. No caso de por algum motivo não consiga concluir a tarefa, o serviço deve armazenar na base de dados um erro na execução do pedido efectuado.

Desta forma não é necessária uma aplicação local intermédia para efectuar a comunicação e armazenamento de dados do recurso, contudo, os *Web services* serão funções bastante complexas e pesadas que necessitarão de conter todas as rotinas necessárias para ligação e comunicação com os dispositivos e rotinas de armazenamento de dados. A possibilidade de falha de transferências de dados ou falha na comunicação é maior e existirá um menor controlo sobre todas as entidades do sistema.

Neste caso o servidor local funciona apenas como uma ponte de ligação entre as aplicações cliente e os recursos, uma vez que este apenas disponibiliza os *Web services* para as aplicações cliente, estabelece a ligação física com os recursos e contém a(s) base(s) de dados onde serão armazenados todos os dados e informações.

Desenvolver os *Web services* permitindo que a base dados sirva apenas para armazenamento dos dados é viável e não requer que a aplicação cliente efectue um "pull" para saber os dados da ordem efectuada. Ao efectuar uma ordem, o resultado que o cliente obtém é o valor/dado pedido e não apenas uma indicação de que o servidor local conseguiu concluir a ordem com sucesso ou não. Todavia, é possível

perder de dados e que sejam gerados alguns conflitos entre aplicações, como consequência de algumas falhas de comunicação ou mesmo tempos elevados para efectuar determinadas tarefas.

Hipótese 2

Uma segunda solução a propor consiste numa arquitectura com duas camadas distintas, uma composta pela aplicação cliente e *Web services*, que vão enviar e obter informações a partir da base de dados. A outra camada será constituída pela aplicação local e o recurso, sendo que a aplicação local obtém os dados das ordens a realizar a partir da base de dados e contém todas as rotinas necessárias para que possa comunicar com o recurso, executando as ordens/operações pretendidas pelo cliente.

No servidor local teremos a aplicação local, a ligação física ao recurso, a base de dados e os *Web services* disponíveis para as aplicações cliente, como ilustra a Figura 19.

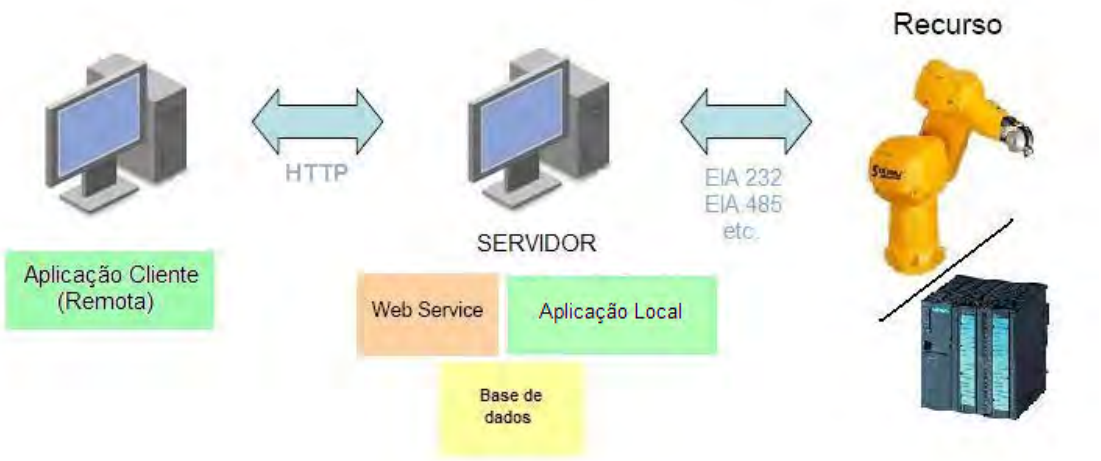


Figura 19 - Esquema da arquitectura de duas camadas proposta na hipótese 2.

A aplicação local para além de permitir a monitorização e controlo remoto do recurso anula a perda de dados e informações, pois todas as trocas de informação, com a aplicação cliente, podem ser armazenadas na base de dados e efectuadas a partir daí.

A aplicação cliente recorre aos serviços disponíveis para enviar os pedidos/ordens para a base de dados no servidor local, via *TCP/IP*. A base de dados será constantemente verificada pela aplicação local para confirmar se há novos pedidos/ordens para efectuar e se os dados enviados estiverem correctos, a aplicação local efectuará o pedido/ordem comunicando com o recurso através das rotinas de comunicação, implementadas de acordo com o protocolo de comunicação do recurso. Após efectuada a comunicação com o recurso as informações serão armazenadas na base de dados e a aplicação cliente terá que executar um serviço do tipo "*pull*" do resultado, para tomar conhecimento do resultado do último pedido/ordem efectuado, ou seja, efectuará uma leitura na base de dados.

A aplicação local, para além de permitir a interacção entre uma ou mais aplicações cliente remotas com um recurso ou vários, normalmente, contém módulos que permitem efectuar o controlo e monitorização local do recurso. Neste sentido, na aplicação local deverá ser possível verificar e alterar estados e valores, para além de se poder executar determinadas operações, que deverão ser as mesmas possíveis através dos *Web services*. Desta forma, localmente será também viável identificar problemas e gerir processos e estados.

A aplicação local deverá ser flexível, permitindo ser parametrizada, de forma que para os *Web services* relativos a processos/tarefas para os quais o cliente não necessite saber quais as variáveis e parâmetros necessários, sejam válidos mesmo que se trate de um equipamento diferente ou de um *PLC* diferente. Com esta potencialidade será possível criar *Web services* globais e reutilizáveis.

A proposta de arquitectura aparenta ser bastante viável, considerando que a perda de dados e informação entre aplicações é difícil visto que a troca de dados entre aplicações passa pela base de dados, o que implica que a informação fique armazenada. A aplicação local ao tratar os dados e gerir as comunicações com os recursos possibilita que os *Web services* sejam mais simples, fiáveis e menos complexos. A aplicação local acrescenta outra valência, realizar a monitorização e controlo local do recurso.

Hipótese 3

A última solução que se propõe implementar consiste numa arquitectura com três camadas distintas, como se pode analisar na Figura 20. A primeira camada é composta pela aplicação cliente e os *Web services*, que vão enviar e obter dados a partir da base de dados. A segunda camada caracteriza-se por uma aplicação servidor que a partir das bases de dados recolhe a informação, trabalha-a e organiza-a. A última camada é a aplicação local que obtém os dados (previamente tratados), a partir de uma base de dados, das ordens/operações a realizar e contém todas as rotinas necessárias para que possa comunicar com o recurso.

A aplicação local executa as ordens/operações pretendidas pelo cliente e envia os resultados para a base dados para que a aplicação servidora os processe e coloque na base de dados para a aplicação cliente consultar.

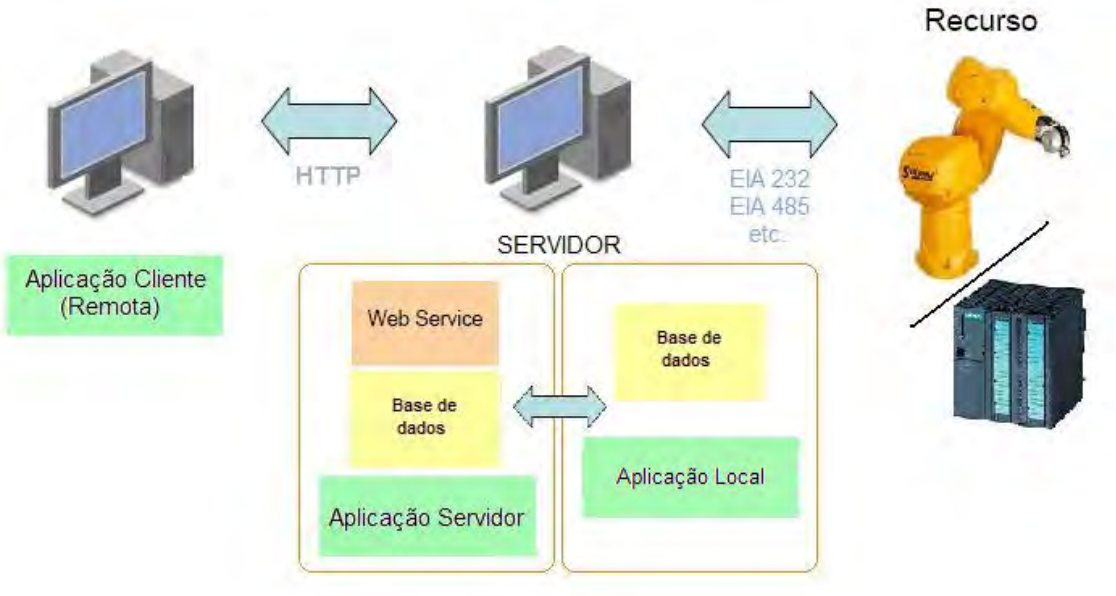


Figura 20 - Esquema da arquitectura de três camadas proposta na hipotese 3.

A aplicação local pode estar no mesmo computador/servidor que a aplicação servidor e os *Web services* ou, poderá estar num computador diferente, associado aos recursos, justificando assim a possibilidade de existirem várias aplicações locais.

Esta proposta é semelhante à anteriormente apresentada, no entanto, incorpora mais uma etapa entre aplicação cliente e o recurso que é realizada pela aplicação servidor, cuja função é apenas tratar e preparar os dados para facilitar a comunicação entre aplicação cliente e a aplicação local e recurso. Desta forma, há maior organização no armazenamento de dados, sendo possível separá-los dos dados necessários e relativos às comunicações efectuadas com os recursos.

A aplicação local, à semelhança da aplicação local da proposta anterior, contém a ligação física e gere as comunicações e ligações com os recursos, contendo ainda os módulos cruciais para efectuar a monitorização e controlo local do recurso.

A solução permite maior e melhor organização, no entanto, também eleva a complexidade do sistema, tendo por base de comparação a solução anterior, com esta passam a existir duas entidades (aplicações) distintas para efectuar o mesmo trabalho, que apenas uma aplicação pode realizar. Com esta solução complica-se e torna-se mais trabalhoso adaptar toda a arquitectura a recursos diferentes ou mesmo a *PLC's* distintos, ou seja, diminuindo a flexibilidade do sistema.

A proposta vale pela organização de dados e informações que se consegue, uma vez que passa-se a ter bases de dados ou esquemas diferentes, de uma base de dados, para armazenar os dados e informações relativas à aplicação cliente e os dados relativos às ligações. Todavia, numa perspectiva inicial não parece ser a hipótese que melhor se enquadra com a solução pretendida, pelo facto de ter vários pontos de

tratamento e transmissão de dados, o que pode suscitar problemas ou mesmo perda de informação entre pontos/aplicações.

A primeira hipótese implica a criação e de *Web services,* programaticamente, muito extensos e pesados, o que leva a tempos de troca de dados elevados e a possibilidade de perdas de dados. A adaptação e integração de novas equipamentos poderá ser mais difícil, devido á necessidade de incluir nos *Web services* as rotinas necessárias para efectuar a comunicação com os equipamentos.

A segunda hipótese apresenta-se, praticamente, como uma evolução da anterior, pois introduz na arquitectura, uma aplicação local que interpreta os pedidos provenientes das aplicações cliente e efectua a comunicação com os equipamentos, colocando os dados provenientes dos equipamentos na base de dados, para consulta. Desta forma os *Web services* passam a ser apenas uma ferramenta de troca de dados, sendo mais fáceis de integrar e interagir. Todos os dados passam também a ser armazenados.

A terceira hipótese é equivalente á segunda, sendo que passa a ter uma arquitectura que permite maior organização de dados devido à interacção entre aplicação servidor e local. No entanto são utilizados dois elementos, o que representa maior complexidade, quando todo o processamento pode ser efectuado apenas por uma aplicação local.

Após uma primeira análise das várias hipóteses propostas e de efectuados alguns testes, a aplicação que se mostrou mais sensata e que mais se identifica com a solução para o problema proposto é a hipótese 2, sendo que das três é a que apresenta mais garantias do bom funcionamento de todo o sistema e com alguma simplicidade.

3.2 Equipamentos

Numa primeira fase verificou-se em vários ambientes industriais e também em ambientes académicos, quais as marcas de *PLC's* e outros equipamentos mais comuns. A atenção recaiu sobre os *PLC's*, equipamentos muito versáteis que podem ser encontrados nas mais variadas aplicações e recursos, sendo mesmo o tipo de equipamento mais comum em muitas empresas.

A facilidade de acesso aos equipamentos foi o segundo parâmetro a ter em conta, impondo-se a extrema necessidade de ter acesso aos equipamentos para poder testar o sistema implementado.

Algumas das marcas que mais se destacam, relativamente aos *PLC's,* são: *Siemens*, *Mitsubishi*, *Telemecanique, Beckhoff, Omron e Fatek*. Posto isto, e considerando a facilidade de acesso aos equipamentos, foram então seleccionados as seguintes marcas:

- *Siemens*
- *Mitsubishi*
- *Omron*

No que respeita à *Siemens* foram consideradas e ensaiadas as séries S7 300 e S7 200, para as quais o protocolo de comunicação é diferente, como diferem também algumas áreas de memória.

Para a *Mitsubshi* a solução encontrada permite a comunicação com uma vasta gama de séries:

- *FX*
- *A*
- *QnA*
- *QQ*
- *QA*

Dentro destes modelos, a solução implementada foi ensaiada com os modelos FX2N e com o modelo AN1S.

A *Omron* tem uma vasta gama de *PLC's, apesar disso,* os protocolos de comunicação não são os mesmos para todas as séries. O *PLC Omron* CPM2A, foi o utilizado para testar o sistema implementado, foi necessário verificar qual o protocolo mais indicado para este modelo, tendo sido seleccionado o *Hostlink* comandos *C-mode* que permite a comunicação com os seguintes modelos:

- *CPM*
- *CJ*
- *CS*

Para além dos *PLC's,* também comando numérico Fagor 8050 foi avaliado, por ser um equipamento com o qual já havia trabalhado e por estar disponível nas instalações do Departamento de Engenharia Mecânica. Um outro pressuposto ao considerar o Fargor 8050 foi o de testar a versatilidade do sistema proposto.

No anexo A4, é possível conhecer melhor as características dos equipamentos testados e ficar também a conhecer as áreas de memória disponíveis em cada marca.

3.3 Base de dados

A base de dados apresenta-se como uma componente essencial na arquitectura proposta sendo este o meio da troca de dados entre aplicações. Com a base de dados, Com a base de dados, torna-se possível a consulta a todos os dados e em qualquer momento, desde dados relativos a pedidos/ordens feitas por aplicações cliente até dados referentes às comunicações e ligações com os recursos.

A selecção do sistema de gestão de base de dados mais adequado e compatível com o pretendido, mereceu alguma pesquisa, no sentido em que o que se pretende é que a base de dados seja flexível, de fácil integração com os *software*s e *Web sevices* e de fácil acesso.

Numa primeira pesquisa, os sistemas de gestão de bases de dados (SGBDs) que se mostram possíveis para a solução em questão foram o *Microsoft* Access, o *Microsoft* SQL e o Oracle - sistemas de gestão de bases dados bem conhecidos e com créditos dados. Qualquer um dos sistemas de base de dados

poderia ser utilizado, no entanto, o objectivo é a utilização de uma base de dados *freeware* e que não implique encargos com licenças.

As SGBDs freeware mais cotadas são o *MySQL* e a PostgreSQL, sendo o *MySQL* a mais conhecida e a mais usada em todo o mundo, mesmo em companhias de relevância mundial como a NASA.

A SGBD *MySQL* enquadra-se perfeitamente com o pretendido, uma vez que para além de ser freeware, tem as características necessárias para o sistema proposto, é muito fácil de integrar numa aplicação e é muito rápida, o que se reflecte como uma mais-valia para diminuir o atraso na comunicação de dados desde o pedido da aplicação cliente até receber a resposta.

O *MySQL* é um SGBD que é optimizado para aplicações *Web* e é amplamente utilizado na Internet. É comum encontrar serviços de armazenamento de *sites* que oferecem o *MySQL* e a linguagem *PHP*, porque ambos funcionam bem em conjunto [Wikipedia, 2009b].

Um outro ponto a favor do *MySQL* é a sua compatibilidade com a maioria dos sistemas operativos, *Linux*, *Mac OS X*, *Windows*, etc.

Entre as características técnicas do SGBD *MySQL* estão [Alecrim, 2006]:

- Alta compatibilidade com linguagens como *PHP, Java, Python, C#, Ruby e C/C++, VB.Net*;
- Portabilidade (suporta praticamente qualquer plataforma actual);
- Baixa exigência de processamento (em comparação como outros SGBD);
- Recursos como *transactions* (transacções), ligação segura, indexação de campos de texto, replicação, etc;
- Instruções em *SQL*;
- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de *hardware*;
- Facilidade de uso, implementação e integração;
- Software Livre;
- Interfaces gráficas "*MySQL Toolkit*" de fácil utilização cedidos pela *MySQL* Inc.

Estrutura da base de dados do *MySQL* a criar

A base de dados será composta de forma permitir o armazenamento eficiente dos dados, resultando numa estrutura compreensível e de fácil acesso aos diferentes dados.

Com o objectivo de evitar a confusão entre dados de diferentes recursos, no SGBD serão criados esquemas (bases de dados) distintos para cada recurso, o que implica que as tabelas de armazenamento dos dados figurem agrupadas por esquemas relativos a um recurso em específico.

De seguida pode ver-se a estrutura de um esquema de dados do *MySQL* para uma mesa de circulação de paletes, um esquema denominado mesa_circular que apresenta as várias tabelas necessárias para o armazenamento de dados e comunicação de aplicações cliente com a aplicação local.

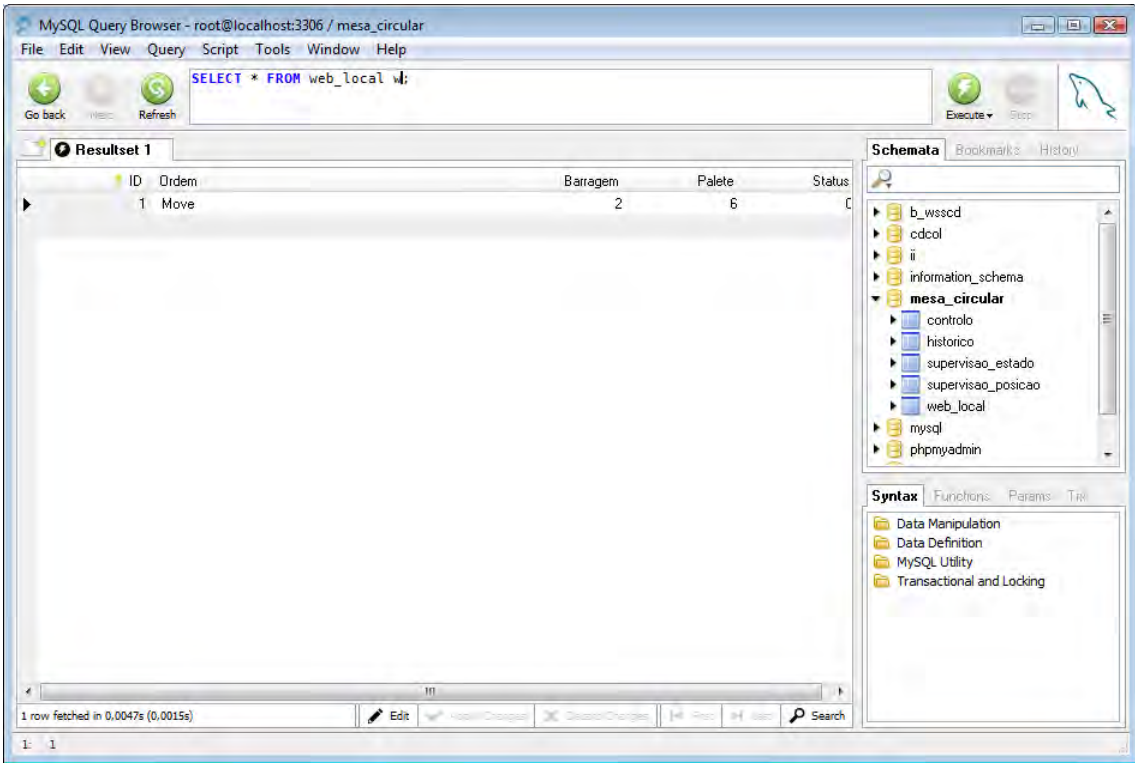


Figura 21 - Estrutura da base de dados para controlo local e remoto de uma mesa de circulação de paletes.

Vários esquemas serão criados e cada um deles identificado pelo nome do *PLC* a controlar, de maneira a que se possa desenvolver uma estrutura para vários *PLC's*, criando interfaces próprias de comunicação.

Cada esquema será composto por várias tabelas, que terão uma função específica.

Tabelas:

- Comunicação e troca de dados entre aplicação cliente e aplicação local (*com*).
- Tabela de dados de resposta relativos a ordens/pedidos para aplicação cliente executar um "*pull*" dos dados relativos à última ordem/pedido (*resp*).
- Tabela de histórico de pedidos da aplicação cliente (*hist_ordens*).
- Tabela de histórico de dados e informações (*hist_values*).
- Tabela de informações sobre variáveis para as quais será pedida actualização contínua (*subscribed_vars*).
- Tabela do estado ou valores das variáveis, tendo sido solicitada a actualização contínua, para ser consultada pela aplicação cliente (*subs_resp*).

Para todos os recursos ou equipamentos existirá um esquema diferente na SGBD, com as tabelas supra mencionadas, exactamente com a mesma nomenclatura e estrutura. Assim será possível criar uma norma de programação, tornando exequível adaptar ou acrescentar um novo recurso, sendo que no sistema de gestão dos dados a única diferença será o nome do esquema respectivo ao recurso.

Os dados podem desta forma ser organizados por recurso ou equipamento, evitando perda e troca de informações. Por outro lado, há que sublinhar que será sempre possível consultar toda a informação relativa às acções efectuadas com solicitação de aplicações cliente, ou servidor.

3.4 Web services

Após a análise de vários *softwares SCADA* e *OPC* Servers pode-se determinar as principais necessidades que potenciem a criação de um conjunto de serviços com características semelhantes, que permitam o acesso aos vários tipos de variáveis e estados, como estados do *PLC* e ligação e ainda o acesso a históricos de dados e estados…

As especificações *OPC* também são muito importantes para definir os *Web services*. A especificação *OPC* DA e a análise de vários servidores *OPC* DA apresentam-se relevantes, tendo em conta que este tipo de servidor exerce a função que se pretende que os *Web services* desempenhem, mas localmente, ou, em alguns casos, remotamente, para além de disponibilizarem as funções necessárias para o desenvolvimento de uma aplicação cliente para supervisão e controlo de recursos. Os *Web services* também permitem uma abstracção dos protocolos de comunicação com os dispositivos de controlo dos recursos.

Remontando ao capítulo 2.3.1, os servidores *OPC* DA permitem a criação de *Tags*, indexadas a variáveis ou blocos de variáveis, às quais poderemos ter acesso ou mesmo alterar o seu estado ou valor na aplicação cliente, através de funções criadas para esse efeito.

A *OPC* UA assume também uma importância notória, uma vez que agrega todas as outras especificações, tentando definir uma norma mais organizada, coerente e funcional. A especificação *OPC UA* goza de todas as valências das especificações *OPC*.

Em analogia com um servidor *OPC*, e mesmo com um *software SCADA*, os *Web services* serão definidos em função de vários parâmetros, devendo disponibilizar funções necessárias para permitir a leitura de variáveis ou blocos de variáveis e também tornar praticável a edição do estado ou valor das variáveis, de forma fácil, intuitiva e com o menor número de dados e parâmetros possíveis, de forma a evitar a complexidade dos *Web services*.

Os *Web services* a desenvolver devem poder executar as seguintes funções:

- Ler ou escrever dados das áreas de dados dos *PLC*;
    1. Leitura e escrita de variáveis digitais;
    2. Leitura e escrita de variáveis analógicas.

3. Leitura de temporizadores e contadores.

- Leitura de temporizadores Consulta de históricos;
- Pedir actualizações contínuas do estado de variáveis;
- Verificar o estado da ligação ao *PLC*.

Para criar todos os *Web services* que permitam realizar as tarefas supra indicadas será necessário, contudo, ter especial atenção à especificação *OPC XML* DA da *OPC Foundation*, dado que como é descrito no capítulo 2.3.2, esta especificação é própria para os *Web services* , desta forma impera-se a sua filosofia para os *Web services* a desenvolver.

Os *Web services* terão por base uma estrutura cliente servidor, de acordo com a especificação *OPC XML* DA e toda a estrutura a desenvolver deverá estar o mais próximo possível dessa especificação.

De acordo com a *OPC XML* DA, será estabelecida uma subscrição do tipo "*polled-pull*", ou seja, o cliente estabelece um "contracto", uma falsa ligação com o servidor, isto porque a comunicação entre os dois não é permanente, nem tão pouco real e directa. O cliente inicia a subscrição e acorda a execução de pedidos periódicos para a actualização dos dados, ou mesmo, vai pedindo a actualização de determinados dados ou parâmetros. No passo seguinte, deverá executar um "*pull*" dos dados, para obter a informação de que necessita. Na Figura abaixo pode ver-se, resumidamente, um diagrama de interacção de um *Web service*, desde o pedido inicial até à obtenção da resposta. A Figura 22 esquematiza a interacção entre aplicação cliente e o equipamento, através dos *Web services*.

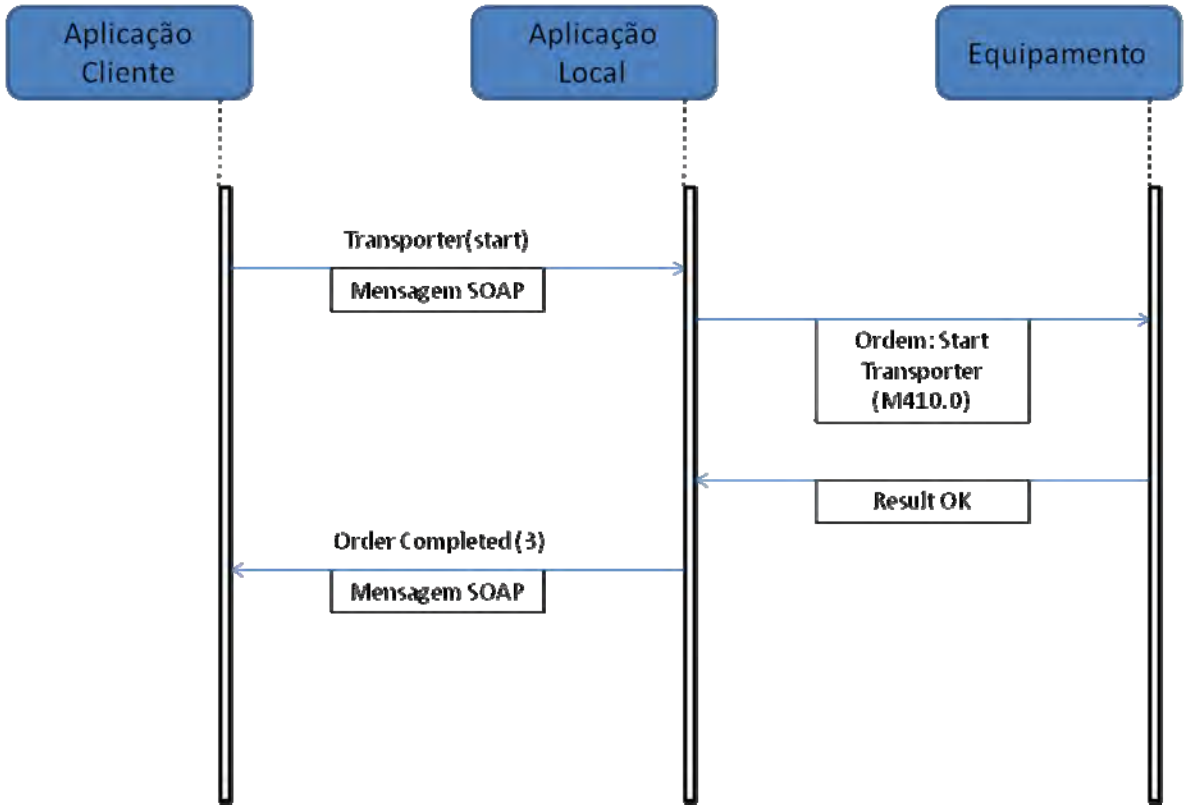


Figura 22 - Exemplo de diagrama de interacção de um *Web service*.

A especificação *OPC XML* DA não e única a ser tida em conta, ou seja, serão criados outro tipo de serviços, mais rígidos e mais específicos, no entanto, recicláveis ou reutilizáveis, enquanto os *OPC XML DA* são de mais baixo nível, generalistas, comuns entre vários servidores *OPC* e que obrigam a um conhecimento dos recursos e programação dos mesmos, ou pelo menos conhecer a estrutura do programa.

Os serviços a criar são definidos para um recurso específico, mas que podem ser utilizados facilmente num recurso diferente, sendo necessário alterar apenas alguns parâmetros na aplicação local e mantendo o *Web service* igual e com os mesmos parâmetros. Desta forma, serão serviços relativos a processos e ou tarefas que um recurso ou equipamento poderá fazer ou fornecer, sem a necessidade de conhecer quais a variáveis a que se deve aceder para executar o pretendido.

Este tipo de serviços poderá ser desenvolvido ou adaptado em qualquer altura e de acordo com um recurso específico, todavia, serão propostos alguns serviços que se apresentarão como os mais apropriados para diferentes recursos e processos, como são exemplos:

- *Cycle* – para iniciar e parar, respectivamente, ciclos automáticos num recurso;
- *Transporter*, com os parâmetros *start*, *stop*, *back* e *front* e indicação do número do transportador, para o caso de existirem vários – serviço apropriado para recursos onde existam tapetes de transporte, transportadores, linhas de alimentação e/ou descarga;

- *Product*, com os parâmetros *OK, NOK* e *ALL* – serviço para retribuir a quantidade de peças conformes ou não conformes ou totais produzidas por um recurso.
- *Movep2p*, com parâmetros de entrada de pontos de início e fim (P1 e P2) – serviço que pode facilmente ser adaptado a pórticos de transferências de peça, mesas de transferência ou mesmo a linhas de transportadores.
- *Rafale* – serviço que permitirá que num recurso flexível (onde poderão ser produzidos ou testados vários tipos de peça) a troca de peças a produzir/testar. Este serviço terá como parâmetro de entrada a identificação da peça a produzir/testar. Exemplos de utilização: controlo e medição de séries de peças, controlo de pinhões, eixos, máquinas de corte de perfis, perfiladoras, etc.
- *Connect* – pedido para verificar o estado da ligação entre aplicação local e o *PLC*.

Os referidos serviços são alguns dos que se poderão criar e que se mostram bastante úteis e aplicáveis para os mais variados recursos e processos.

Todos os serviços criados deverão ter indicação da sua função, para além de descrever quais os parâmetros de entrada, para que o utilizador tenha conhecimento da função a executar e as possibilidades existentes, evitando assim a execução de acções indesejadas.

Durante o processo de desenvolvimento poderão e deverão ser criados mais serviços para aumentar o leque de serviços e a utilidade, eficiência e versatilidade do sistema a criar.

3.5 Aplicação local

Como já foi referido, para permitir que um conjunto de serviços *Web* possam monitorizar ou controlar um equipamento, ou seja, a viabilidade da realização de sistemas *SCADA* remotos, é vital a existência de uma aplicação local. De acordo com a arquitectura seleccionada, será necessária uma aplicação que estabeleça a ponte entre aplicações cliente e os equipamentos, neste sentido, a aplicação será desenvolvida em Visual Basic.Net e permitirá a comunicação com os vários dispositivos seleccionados, sendo apenas uma aplicação para gerir os vários *serviços* criados para cada equipamento.

O programa consultará, constantemente, uma tabela de troca de informação com as aplicações cliente, de forma a processar, com o menor atraso possível, os pedidos das aplicações cliente. A aplicação local toma conhecimento de um pedido e inicia o processamento do mesmo e actualiza a base de dados para informar que esta se encontra ocupada. Após processar o pedido deverá actualizar, a base de dados, com o resultado do pedido, ou seja, se foi concluído ou se houve um erro.

Com esta gestão, torna-se possível gerir os pedidos contínuos dos clientes que não poderão sobrepor um pedido a um pedido em execução, podendo efectuar o pedido logo que a aplicação local esteja disponível.

O programa deverá ter informação actualizada sobre o estado do equipamento, de forma a manter as aplicações cliente informadas do estado do equipamento, ou seja, mediante o estado do equipamento a aplicação poderá ou não fazer determinado tipo de movimentos. Por exemplo, se um equipamento estiver em ciclo automático, não será permitido a aplicações cliente efectuar movimentos ou alterações no ciclo.
O programa será uma pequena aplicação em ambiente *Windows* e é caracterizada por:

- Interface gráfica simples e intuitiva, possibilitando a verificação de todas as tabelas da base de dados. A interface é orientada a objectos, permitindo a sua selecção em função do tipo de equipamento que se perspectiva.
- A tabela de troca de informação dos diferentes equipamentos deverá ser consultada constantemente, a fim de garantir um rápido processamento e transmissão dos dados
- Permitirá a supervisão e controlo local, de forma simples e rápida, mas operação a operação.

A programação será estruturada de forma a permitir a sua fácil adaptação e permitir que de forma simples se adicione e crie toda a estrutura para um novo equipamento. Com estes pressupostos, o programa será muito centrado em "*arrays*", matrizes, funções e com grande recurso a variáveis públicas.

O programa deverá estar devidamente estruturado e dividido por blocos, para permitir aceder facilmente a informação e facilitar a edição. Com vista a viabilizar a comunicação com os equipamentos, recorre-se ao controlador da porta série, uma vez que, apesar da comunicação série ser de certa maneira limitada é, no entanto, muito versátil e simples de utilizar, pelo que é suficiente uma porta série no computador e um cabo de comunicação.

As rotinas de comunicação relacionam-se directamente com o CPU (Unidade de Processamento Central) dos *PLC's* sem que seja necessário realizar programação adicional nos *PLC* ou adicionar *hardware. A*s rotinas de comunicação deverão estar reunidas em módulos separados para cada equipamento.

4 Implementação

Neste capítulo pretende-se descrever todos os passos que foram necessários para implementar o sistema de *Web services* para aplicações *SCADA*, um sistema que possibilita a aplicações cliente remotas, ou apenas através de um *browser*, a monitorização e controlo remoto de recursos ou aplicações controlados por *PLC's*.

A concretização do sistema carece de uma plataforma de comunicação com os *PLC's*, que não exija *hardware* ou *software* adicional, sendo que para isso foi levada a cabo uma pesquisa para desenvolver os protocolos necessários à comunicação com os *CPU's* dos recursos. Seguiu-se a implementação de pequenas aplicações para ensaiar a comunicação, a troca de dados com os recursos e a leitura e escrita da base de dados. Foram desenvolvidos alguns *Web services* para determinar qual a estrutura/arquitectura a implementar e a que se apresentou como mais prática, viável e funcional foi a descrita na hipótese 2 do capítulo 3.1.

A arquitectura proposta foi implementada, tendo-se criado uma pequena aplicação que funciona como aplicação local para os vários equipamentos propostos, para além de se ter desenvolvido os *Web services* independentes para cada equipamento.

Todo o trabalho foi desenvolvido numa perspectiva de *Round-Trip Engineering*, que consiste num misto de *Reverse Engineering* e *Forward Engineering*, o que se traduz num trabalho concluído através de pesquisa, projecto e implementação, mas também através de estudo e adaptação de soluções existentes, de forma a completar um sistema orientado a serviços que permita soluções *SCADA* via *Web*.

Todos os pontos descritos são devidamente analisados de seguida.

4.1 Arquitectura

A arquitectura implementada é orientada a serviços e permite aceder via *Web* aos vários dispositivos seleccionados. Para validar a arquitectura seleccionada será necessário implementar os vários elementos indicados - o servidor local com a aplicação local, a base de dados no SGBD seleccionado e criar os *Web services*.

No servidor local teremos a aplicação local, a ligação física ao recurso, a base de dados e os *Web services* disponíveis para as aplicações cliente. A aplicação local, para além de permitir a monitorização e controlo remoto do recurso, praticamente anula a perda de dados e informações, pelo facto de que todas as

trocas de informação, com a aplicação cliente, são armazenadas na base de dados e são efectuadas a partir da base de dados, havendo sempre um registo.

Conforme ilustra a Figura 23, a aplicação cliente recorre aos serviços disponíveis para enviar os pedidos para a base de dados no servidor local, através dos *Web services,* estes, por sua vez, utilizam uma ligação TCP/IP para interagir com a *SGBD MySQL*. A base de dados é constantemente analisada pela aplicação local, por meio de uma ligação *TCP/IP*, para verificar se há novos pedidos/ordens a efectuar. Se os dados enviados estiverem correctos, a aplicação local efectuará o pedido/ordem comunicando com o recurso através das rotinas de comunicação implementadas de acordo com o protocolo de comunicação do recurso.

Figura 23 - Esquema da arquitectura implementada.

Efectuada a comunicação com o recurso, as informações serão armazenadas na base de dados e a aplicação cliente terá de executar um serviço do tipo "*pull*" do resultado, ou seja, realiza uma leitura na base de dados para verificar o resultado do pedido ou ordem efectuado.

Os elementos da arquitectura em causa podem ser analisados, ressalvando-se a arquitectura funciona como descentralizada e orientada a serviços, que permite a integração de todas as partes criando um sistema distribuído. Os elementos existentes nesta arquitectura são a aplicação cliente, num extremo, o servidor local, que funciona como o "cérebro" do sistema implementado, e no outro extremo os vários equipamentos testados.

No servidor local estão presentes elementos de significativa importância para o bom funcionamento do sistema, como a aplicação local que gere e processa os pedidos das aplicações cliente e efectua as comunicações com os diferentes equipamentos. O Sistema de Gestão de Base de Dados (SGBD), do *MySQL*, é outro elemento crucial, tendo em conta que armazena todos os dados e permite a troca de

informações efectuadas entre aplicações. O último elemento, com igual relevância, são os *Web services* que acessíveis via *Web*, permitem a interacção entre as aplicações, sendo que sem a ausência deste elemento tornaria impraticável a implementação da arquitectura proposta.

4.2 Web services e Aplicação local

A aplicação local e os Web services são os dois parâmetros fundamentais da arquitectura apresentada. A estrutura e modo de funcionamento de cada um, assim como a interacção entre eles através da base de dados são explicados nos pontos seguintes.

4.2.1 Base de dados e sua arquitectura

A base de dados assume relevante, pelo que compreender a sua estrutura é imperativo. A interacção entre aplicações cliente e a aplicação local depende exclusivamente da base de dados, sendo de sublinhar que é essencial que as aplicações estejam em sintonia quanto aos campos que devém consultar ou alterar para realizar a troca de dados, para tal é imprescindível que a base de dados tenha uma arquitectura simples, bem estruturada e compreensível.

O Sistema de Gestão de Base de Dados (SGBD) seleccionado foi o MySQL, como já foi referido (cap. 3.3), tendo sido criados esquemas para cada equipamento, o que permite distinguir os dados de cada equipamento (ver Figura 24), sendo estes identificados pelo equipamento com que estão relacionados:

- B_mits – Esquema para autómato da *Mitsubishi*.
- B_omr – Esquema para autómato da *Omron*.
- B_s7_2 – Esquema para autómato da *Siemens*, série S7 200.
- B_s7_3 – Esquema para autómato da *Siemens*, série S7 300.

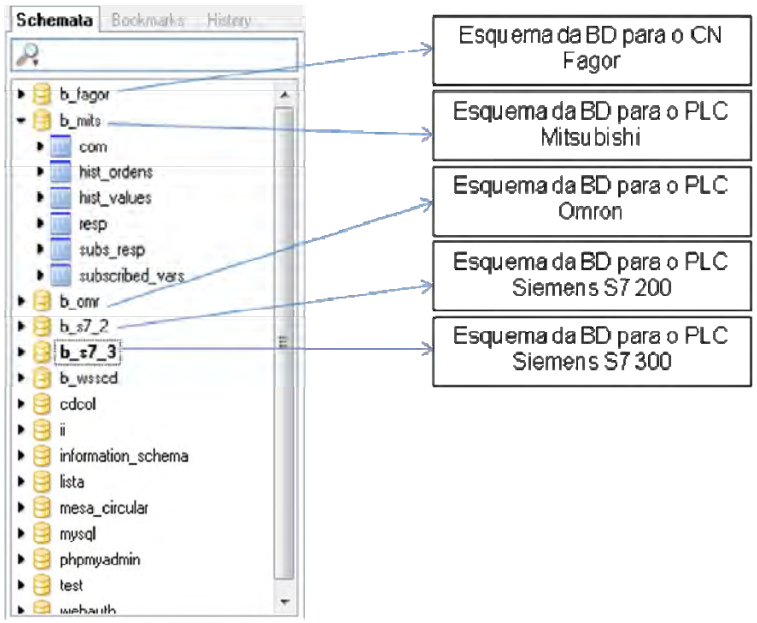


Figura 24 - Esquemas da SGBD *MySQL*.

Os esquemas (Bases de Dados) criados obedecem todos à mesma estrutura, ou seja, são compostos pelo mesmo conjunto de tabelas que têm o mesmo nome e os mesmos campos, o que possibilitou criar uma estrutura para a base dados independente do equipamento e das aplicações que a ela recorrem. Os campos de cada tabela também foram desenvolvidos de forma a satisfazer os mais variados tipos de situações e, seguindo essa estrutura, será sempre fácil identificar as operações e os dados associados.

Como se pode ver na Figura 24, cada esquema é composto por 6 tabelas, cada tabela com uma função específica:

- **com** – tabela de troca de informação entre aplicações cliente e aplicação local. Nesta tabela serão introduzidos todos os pedidos efectuados pelos clientes, sendo que a aplicação local analisa os pedidos e após a conclusão informa sobre o sucesso ou não do pedido. A aplicação local deve manter actualizado o campo relativo ao estado do equipamento.

  Descrição dos vários campos da tabela:

Tabela 3 - Indetificação dos campos da tabela de troca de dados, "com".

Campo	Descrição
ID	Identificação da linha
Ordem	Ordem relativa ao pedido
Var1	1ª Variável ou acção relativa à ordem
Var2	2ª Variável ou acção relativa à ordem
Var3	3ª Variável ou acção relativa à ordem
Value	Valor associado ao pedido/ordem
Status	Estado de processamento de pedidos
Rstate	Estado do Equipamento
Hora	Hora de inserção do pedido
Data	Data de inserção do pedido

- **resp** – Tabela onde são inseridas as respostas aos pedidos efectuados, sendo que a cada tipo de pedido/ordem apenas corresponde uma linha da tabela, não havendo alteração da ordem. A aplicação cliente após efectuar um pedido e este estar concluído, se for o caso, faz o "*pull*" do resultado consultando esta tabela.

Tabela 4 - Indetificação dos campos da tabela de respostas aos pedidos, "resp".

Campo	**Descrição**
ID	Identificação da linha
Ordem	Ordem relativa ao pedido
Var1	1ª Variável ou acção relativa à ordem
Var2	2ª Variável ou acção relativa à ordem
Value	Valor associado ao pedido/ordem
Info	Estado de conclusão do pedido: *Completed (3)* ou *Error (2)*
Hora	Hora da conclusão do pedido pela aplicação local

Data	Data da conclusão do pedido pela aplicação local

- **Hist_ordens** – Tabela de histórico de pedidos em que são armazenados todos os pedidos efectuados por clientes.

Tabela 5 - Indetificação dos campos da tabela de de histórico de pedidos ou ordens.

Campo	Descrição
ID	Identificação da linha
Ordem	Ordem relativa ao pedido
Var1	1ª Variável ou acção relativa à ordem
Var2	2ª Variável ou acção relativa à ordem
Var3	3ª Variável ou acção relativa à ordem
Value	Valor associado ao pedido
Status	Estado de processamento de pedidos
Rstate	Estado do Equipamento
Pedido	Hora de inserção o pedido
Executado	Hora da conclusão do pedido pela aplicação servidor
Data	Data de inserção o pedido

- **Hist_values** – Tabela de histórico de todos os dados e informações de resposta a pedidos.

Tabela 6 - Indetificação dos campos da tabela de histórico das respostas aos pedidos.

Campo	Descrição
ID	Identificação da linha
Ordem	Ordem relativa ao pedido
Var1	1ª Variável ou acção relativa à ordem
Var2	2ª Variável ou acção relativa à ordem
Value	Valor associado ao pedido
Status	Estado de processamento de pedidos
Rstate	Estado do Equipamento
Hora	Hora de inserção do pedido
Data	Data de inserção do pedido

- **Subscribed_vars** – Tabela para inserir variáveis para as quais se pretende actualização contínua do seu estado ou valor. As variáveis são inseridas pelos clientes, assim como podem ser eliminadas.

Tabela 7 - Tabela de subscrição de variáveis para actualização contínua.

Campo	Descrição
Variável	Identificação da variável e seu endereço
Tipo_var	Identificação do tipo de variável (Boleana, Byte, etc.)
Descrição	Breve descrição da variável
Hr_entrada	Hora de inserção do pedido
Dt_entrada	Data da conclusão do pedido

- **Subs_resp** – Nesta tabela constam os estados ou valores actualizados relativos às variáveis da tabela anterior. A tabela é constantemente actualizada pela aplicação local e está directamente relacionada com a tabela "*subscribed_vars*".

Tabela 8 - Tabela com o estado ou valor actualizado das variáveis, para as quais se pretende actualização contínua.

Campo	Descrição
Variável	Identificação da variável e seu endereço
Estado	Estado ou valor da variável
Hora	Hora da última actualização do estado da variável
Data	Data da última actualização do estado da variável

4.2.2 Aplicação local

Aplicação local, como já foi mencionado, foi desenvolvida em *Visual Basic.Net (Visual Studio 2005)* e consiste numa pequena aplicação *Windows* orientada a "objectos". A aplicação tem uma interface muito simples e muito intuitiva, sendo muito fácil interagir com a aplicação.

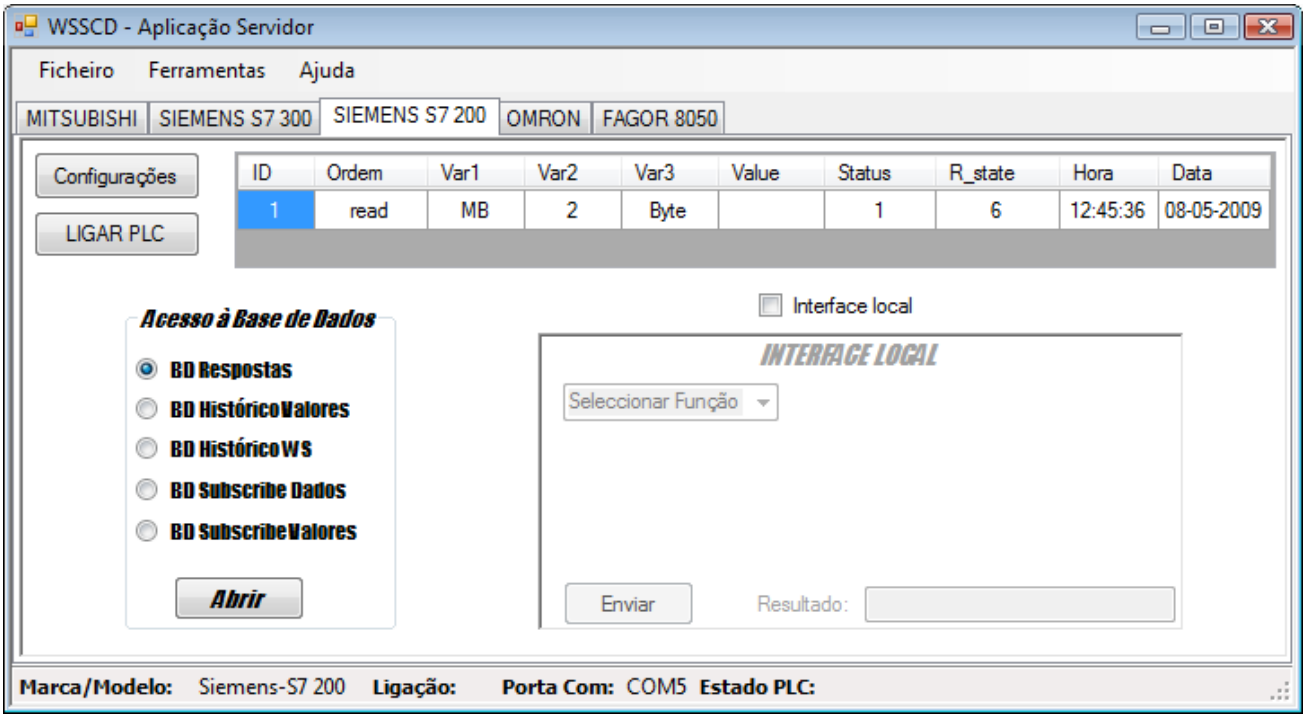


Figura 25 - Interface gráfica da aplicação local.

A Figura 25 ilustra a interface da aplicação local e mostra que é exequível alternar entre páginas (*Tab Pages*), no topo, de acordo com o equipamento pretendido, sendo que em cada página teremos acesso aos seguintes elementos:

- Vista da tabela de troca de dados entre aplicações cliente e a aplicação local, actualizada constantemente (com);
- Acesso às configurações de cada equipamento (Configurações de comunicação);
- Acesso às várias tabelas dos respectivos esquemas das bases de dados (Figura 26).
- Activação e desactivação da interface local e possibilidade de interagir localmente com o respectivo equipamento.

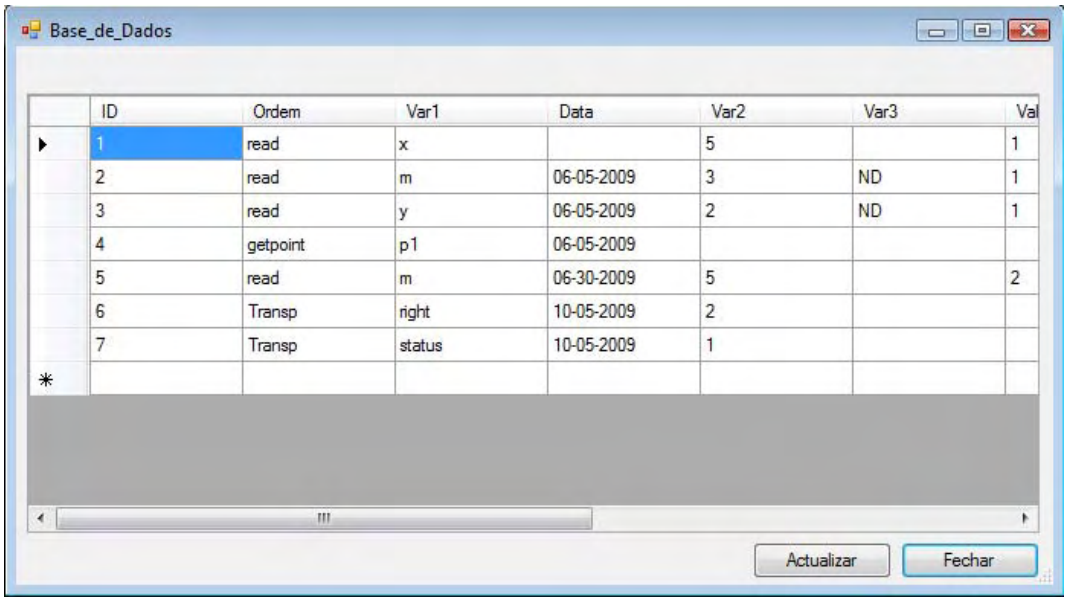


Figura 26 - Exemplo de acesso à Base de Dados, tabela de histórico de pedidos.

A aplicação é simples, sendo a sua principal função trabalhar como uma interface de comunicação entre aplicações clientes, remotas e os equipamentos, para além de possibilitar às aplicações remotas criarem as interfaces desejadas para a monitorização e controlo de um determinado equipamento/processo.

A aplicação, resumidamente, trabalhará segundo o seguinte modelo, Figura 27.

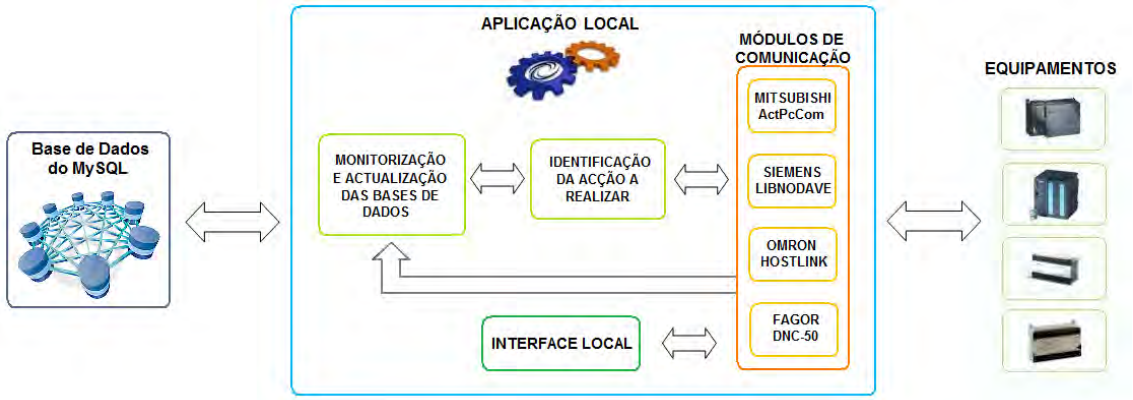


Figura 27 - Modelo de funcionamento da aplicação local.

A aplicação local estará a "vigiar", constantemente a base de dados, mais especificamente a tabela de troca de dados (com) com as aplicações cliente, para que no momento em que o cliente remoto insira um novo pedido, a aplicação local inicie o processamento dos dados e as devidas acções, a fim de corresponder ao pedido efectuado no menor tempo possível.

Caso o pedido efectuado tenha como pressuposto alterar o estado ou valor de uma variável, ou executar uma ordem no equipamento, a aplicação local envia a informação para o equipamento e se for bem sucedida esta aplicação deverá actualizar um campo na base de dados de que o pedido foi efectuado com sucesso. Caso exista, por alguma razão, um erro ou impossibilidade de executar o pedido, a base de dados ser actualizada com a informação de que ocorreu um erro.

Se o pedido feito por uma aplicação cliente, implicar retorno de informação, tal como o estado de uma variável, a aplicação local, após receber os dados do equipamento, deverá actualizar o campo na base de dados relativo ao estado dos pedidos, para além de colocar os dados relativos ao pedido efectuado numa tabela (*resp*) destinada a receber os dados relativos ao último pedido efectuado, que será posteriormente consultada pela aplicação cliente.

Os *Web Services*, ao serem invocados, vão criando um histórico de todos os pedidos dos clientes, guardando no servidor os dados numa tabela denominada "*hist_ordens*".

O programa encontra-se organizado em vários módulos, interfaces (*forms) e user controls*, de maneira que em futuras alterações ou integração de novos dispositivos, a execução seja fácil.

A Figura 28 apresenta todos os componentes do programa na árvore do *Solution Explorer, onde se* podem identificar todos os módulos relativos às rotinas de comunicação, de funções necessárias para a implementação do programa de forma flexível e estruturada, módulos de criação de variáveis, os *forms* de apresentação das tabelas das bases de dados, os *forms* das definições de cada equipamento e o *form* principal onde se encontram todos os comandos possíveis.

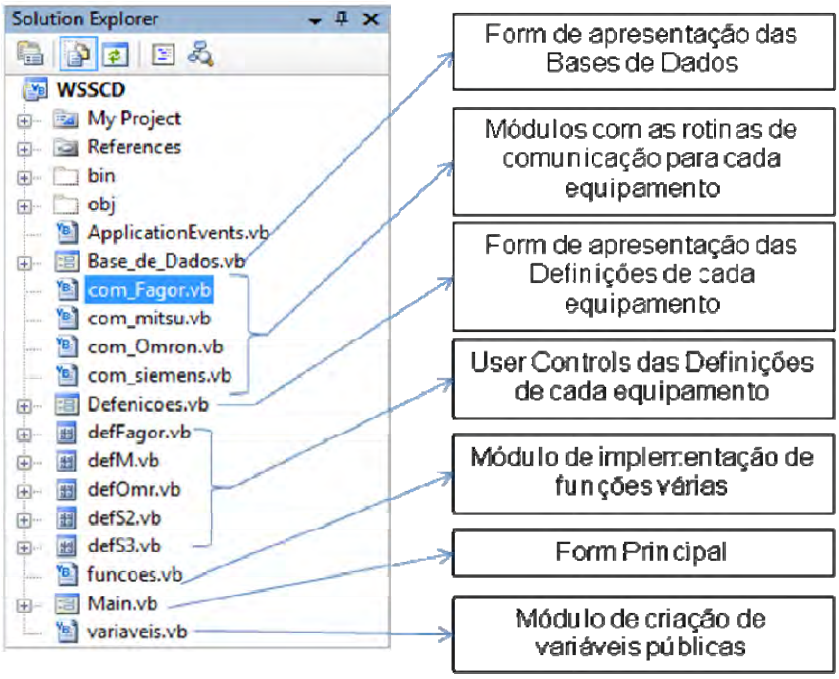


Figura 28 - Árvore do *Solution Explorer* do programa (aplicação local), com todos os componentes criados.

Na árvore do *Solution Explorer* pode-se identificar também as referências (Figura 29) necessárias para o bom funcionamento do programa, desde a comunicação com os equipamentos à comunicação com a base de dados do *MySQL*.

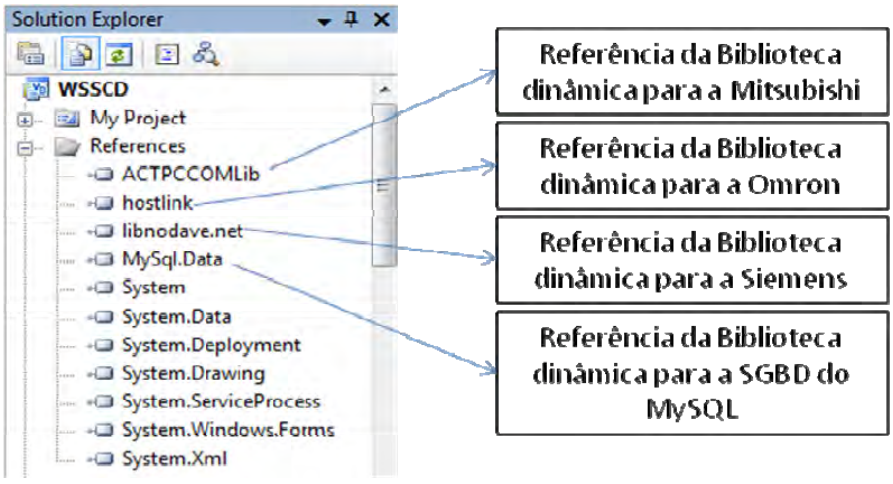


Figura 29 - Solution Explorer do programa, com as referências necessárias.

A aplicação desenvolvida mostrou-se fiável e capaz de interagir com todos os esquemas da base de dados e com os equipamentos, por forma a permitir o correcto funcionamento do sistema implementado e

com pequeno atraso, com objectivo de tentar que nas aplicações cliente os dados estejam muito próximo de dados em tempo real.

4.2.2.1 Implementação de bibliotecas dinâmicas e módulos de comunicação

A comunicação com os dispositivos (autómatos, CN's, etc.) revela-se de extrema importância, sendo que se pretende aceder a todo o tipo de informação e mesmo alterar os valores de variáveis e estados dos equipamentos. A ambição é efectuar a comunicação de forma autónoma, o que não implica o recurso a *software*s proprietários. Um outro objectivo é evitar o recurso a *software*s comerciais, mesmo que estes permitam uma enorme flexibilidade, mas que se reflecte em custos adicionais, quando o objectivo é criar uma aplicação com o menor custo possível.

O *hardware* assume também significativa relevância*,* uma vez que cria um sistema que possibilita o acesso à informação via *Web,* mas que não recorre a *hardware* adicional, para além dos dispositivos e de um computador que fará a "ponte" entre o recurso local e uma aplicação cliente.

Neste sentido, foi necessário efectuar uma pesquisa intensiva para encontrar formas de comunicar directamente com os equipamentos, designadamente com o *CPU* dos dispositivos.

Equipamentos seleccionados:

- Autómato *Siemens* S7 série 300 e 400.
- Autómato *Siemens* S7 série 200.
- Autómatos *Omron* (vasta gama).
- Autómatos *Mitsubishi* (vasta gama).
- Comando numérico Fagor 8050.

Para os equipamentos da *Siemens,* devido à complexidade dos seus protocolos de comunicação e falta de informação sobre estes, foi encontrada uma solução de código aberto que permite criar as funções necessárias para estabelecer a comunicação. A solução denomina-se por *Libnodave* e consiste num projecto desenvolvido por várias pessoas para ultrapassar os protocolos de comunicação da *Siemens*. Esta biblioteca dinâmica possibilita a comunicação com os equipamentos sem que para isso se utilize qualquer *software* da *Siemens*, recorrendo apenas à porta série de um computador e a um cabo para efectuar a ligação ao CPU dos autómatos. No entanto, elevou-se a necessidade de diferenciar os autómatos da série S7 300 e 400 com a série S7 200, pelo facto do protocolo de comunicação para as séries serem diferentes, sendo o MPI (*Multi Point Interface*) para a série S7 300 e 400 e o PPI (*Point to Point Interface*) para a série S7 200.

No anexo A3 é possível analisar o objecto de comunicação *Libnodave*.

A *Mitsubishi* tem uma aplicação (*MX Component)* que pode ser integrada no desenvolvimento de programas, para comunicar através da porta série com vários autómatos da marca, todavia, é algo que implica adquirir e instalar o *software*. Depois de realizada uma pesquisa para ultrapassar este ponto, a opção

a utilização da biblioteca dinâmica *ActPcCom*, que faculta a comunicação e todo o tipo de funções necessárias para a troca de dados com os autómatos da *Mitsubishi*, ou seja, permite efectuar o mesmo tipo de operações que o *software MX* Component, mas sem instalação do mesmo.

Para obter mais informação sobre o objecto *ActPcCom* consultar o anexo A3 em anexo.

A *Omron*, ao contrário da maioria das marcas, disponibiliza muita informação relativamente aos seus equipamentos e protocolos de comunicação. A documentação disponível no *Web* Site da *Omron* foi criteriosamente analisada, tendo-se verificado que estava disponível um protocolo que correspondia às necessidades deste trabalho, em suma, um protocolo que permite de forma simples comunicar e trocar dados com várias séries de autómatos, sendo somente necessário um cabo de comunicação série para comunicar com o computador. O protocolo é conhecido por *Host-link,* um tipo de protocolo de comunicação convencional, onde o *PLC* envia uma resposta consoante os comandos pedidos pelo computador. Os comandos podem ser usados para ler ou escrever dados nas áreas de memória do autómato e controlar algumas operações.

Usando os manuais sobre este tipo de comunicação foi criada uma Biblioteca de Ligação Dinâmica (um ficheiro DLL) para adicionar ao programa e disponibilizar funções, que permitem o mesmo tipo de operações que os objectos anteriormente referidos, tais como:

- Ler ou escrever dados das áreas de dados dos *PLC*;
- Ler ou alterar o estado do CPU;
- Ler e limpar os erros detectados na CPU;
- Ler o modelo do *PLC*.

No anexo A3 o protocolo e a livraria criada são devidamente explicados.

O acesso ao Comando Numérico (CN) Fagor 8050 e à documentação referente ao protocolo de comunicação DNC 50 que possibilita o acesso às variáveis, programas do CN e também a variáveis do *PLC* existente no CN. Foi estudada a hipótese de criar serviços *Web* que permitissem o acesso às variáveis, o que significa aceder ou alterar aos seus valores e estados remotamente e carregar e descarregar os programas do comando numérico localmente.

Em anexo (A3) é possível analisar detalhadamente o protocolo DNC 50.

De sublinhar que todos os objectos/bibliotecas e rotinas utilizadas evitam a necessidade de usar qualquer programa de comunicações no *PLC* para obter os dados e informações pretendidas, bem como alterar estados e variáveis, sem que seja necessário qualquer *hardware* para além dos cabos de comunicação com os respectivos equipamentos.

4.2.3 *Web services*

Os *Web services* são a "ferramenta" que uma aplicação cliente remota poderá utilizar para interagir com um determinado equipamento. Os *Web services* disponibilizam um conjunto de serviços e funções que poderão ser executadas para realizar uma determina acção ou simplesmente aceder a alguns dados, informações, estados e ou valores. Neste sentido, desenvolveu-se um conjunto de serviços, iguais entre equipamentos, ou seja, interoperáveis, que permitem, ao executar esse serviço, realizar a operação pretendida, independentemente do equipamento existente, (dentro da gama de equipamentos seleccionados para este trabalho).

O objectivo passa por colocar ao dispor dos clientes serviços de fácil integração na sua aplicação e que com parâmetros de entrada relativamente simples permitirão executar as acções pretendidas, sem que o gestor da aplicação cliente tenha de conhecer quais os dados a que se deve aceder no equipamento para realizar a acção pretendida.

Outros serviços foram criados, para além do tipo de serviços acima indicados. Os serviços criados são comuns aos disponíveis em qualquer servidor *OPC* e em aplicações *SCADA*, sendo os serviços de leitura e escrita do estado, e valores de variáveis de memória do equipamento e consulta de estados. Este tipo de serviços é também indicado pela especificação *OPC XML DA*, da *OPC Foundation.*

A especificação *OPC XML DA* tem detalhado um conjunto de serviços, para além dos típicos referidos, que permite a actualização contínua dos estados de variáveis de um equipamento, sendo necessário indicar quais as variáveis pretendidas e de seguida apenas requerer o acesso a essa informação.

Serviços criados:

- *Cycle* – serviço que permite verificar se o equipamento está em ciclo automático, iniciar ou terminar um ciclo que esteja em curso. Tem apenas um parâmetro de entrada do tipo *String* e aceita os seguintes valores:
  - *status* – determina se o equipamento está em ciclo ou não.
  - *start* – inicia um novo ciclo.
  - *stop* – terminar o ciclo automático do equipamento.
- *GetStatus* – um serviço que devolve dois índices, em que o primeiro corresponde ao estado do equipamento (modo Manual (0) ou modo Automático (1)). O segundo índice corresponde à existência de defeitos, sendo (0) se não houver defeitos e (1) se existir algum defeito.
- *Movep2p* – serviço apropriado para pórticos de transferência, manipuladores, etc. tendo em conta que permite enviar um pedido para o equipamento se deslocar entre dois pontos distintos. Tem dois parâmetros de entrada, do tipo *String*, relativos ao ponto inicial e final.

- *Product* – serviço relativo a equipamentos de produção de peças, ensaios e montagem, ou de controlo de peças, podendo devolver o número de peças produzidas conformes, não conformes e o número total de peças. O *Product* tem um parâmetro de entrada do tipo *String* aceitando:
    - *OK* – para o número de peças conformes.
    - *NOK* – para o número de peças não conformes.
    - *ALL* – para o total das peças.
- *Rafale* – proporciona a alteração da série/tipo de peças a produzir, em equipamentos flexíveis, onde se podem produzir, ensaiar ou controlar diferentes tipos de peça. Tem um parâmetro de entrada do tipo *String* onde se deve indicar a série a produzir. Um exemplo de aplicação são máquinas perfiladoras, preparadas para produzir diferentes perfis, neste caso a aplicação cliente poderá escolher qual o tipo de perfil a produzir.
- *Transporter* – serviço apropriado para telas transportadoras e todo o tipo de transportadores, pelo motivo de que permite saber qual o estado do transportador ou escolher o sentido de trabalho, possibilitando ainda iniciar ou parar o transportador. Este serviço tem dois parâmetros de entrada. O primeiro, do tipo *String*, permite os seguintes dados de entrada:
    - *status* – saber o sentido e se está em movimento ou não, recebendo como resposta dois índices, o primeiro relativo ao sentido sendo 0 para trás (back) e 1 para a frente (front). O segundo índice será 0 se estiver parado e 1 se estiver em movimento.
    - *back* ou *front* – para definir o sentido do transportador.
    - *start* ou *stop* – para iniciar ou parar o transportador.

O segundo parâmetro é do tipo *Inteiro* e tem como função identificar o número do transportador caso esteja presente uma aplicação com vários transportadores, sendo que se existir apenas um transportador não é necessário preencher este campo.

- *Read_var* – serviço com o qual se pode pedir informação sobre o estado ou valor de uma ou mais variáveis. Por meio deste serviço é possível consultar todo o tipo de variáveis dos equipamentos. O *Read_var* exige um maior número de parâmetros de entrada, contudo, é extremamente útil e funcional. Parâmetros de entrada:
    - Variavel – indicação da área de memória do equipamento (*String*).
    - Endereço – indicação do endereço da respectiva variável (*Integer*)
    - Num_vars – Número de variáveis a ler a partir do endereço indicado, 1 por defeito (*Integer*).
    - Descricao – descrição do tipo da variável, (Boleana, Decimal, etc.), (*String*).
- *Write_var* – permite alterar o estado ou valor de uma variável, tem os mesmos parâmetros de entrada que o serviço "*Read_var*", para além do parâmetro "*Valor*" onde se deve colocar o valor a enviar (*String*).

- *Subscribe_var* – tem como finalidade inserir uma variável para a qual se pretende uma actualização contínua do seu estado ou valor, tendo os mesmos parâmetros de entrada que o serviço "*Read_var*". Este serviço não insere um pedido na tabela "***com***" da base de dados, pelo contrário, insere os dados da variável e a descrição da mesma na tabela "*subscribed_vars"*, que é constantemente monitorizada pela aplicação local, para inserir os valores actualizados das variáveis na tabela "*subs_resp"*.
- *Get_subscribe_response* – consulta a tabela "*subs_resp"* para devolver os valores actualizados das variáveis previamente inseridas, no formato de um *Dataset* (objecto de armazenamento de dados do Visual Basic.Net).
- *Del_subscribed_var* – serviço que permite eliminar uma variável da tabela de actualização contínua, "*subscribed_vars"*, bastando apenas inserir a variável e o respectivo endereço.
- *Connection* – permite consultar o estado da ligação entre a aplicação local e o equipamento
- *Get_response* – serviço muito importante pois permite obter as respostas de pedidos anteriores, devendo ser executado sempre que se faça um pedido, para saber se foi concretizado com sucesso e obter a resposta, se for o caso. Tem um parâmetro de entrada do tipo *String* que será o nome do pedido efectuado, contudo, não é obrigatório, pois por defeito o serviço verifica qual o último pedido efectuado para de seguida devolver a resposta a esse pedido.
  A resposta devolvida tem, sempre, na primeira posição o estado/resultado *(Status)* do serviço *(0, 1, 2 ou 3*), se o pedido for executado com sucesso, seguem-se os parâmetros do pedido e respectiva resposta e hora a que foi concluído, separados por *";"* exemplo: 3;read;I0.2;1;1;15:51:23.
- *Get_hist_WS* – devolve um *Dataset* com o histórico dos pedidos efectuados.
- *Get_hist_values* – devolve um *Dataset* com o histórico dos valores relativos aos pedidos efectuados.

Os serviços criados mostram-se eficazes no que respeita à função que lhes é atribuída, desde que os parâmetros sejam inseridos correctamente.

Os serviços mais flexíveis e que proporcionam acesso a todas as áreas de memória dos equipamentos são os serviços *read* e *write,* para ler e escrever o estado de variáveis, e *subscribe_var* para inserir uma variável para actualização contínua. Estes serviços são seguidos de *get_response,* para obter as respostas aos pedidos de leitura de variáveis, e *get_subs_resp,* para obter os estados actualizados das variáveis inseridas pelo serviço *subscribe_var.* Estes serviços foram implementados tentando que se mantivessem de acordo com a especificação *OPC XML DA* e respeitando a arquitectura cliente servidor.

Os serviços podem, desta forma, ser adicionados com referências *Web* em programas, no capítulo 4.3 será apresentado um exemplo de aplicação cliente que recorre a estes serviços.

As seguintes Figuras ilustram a utilização um *Browser* como cliente destes serviços.

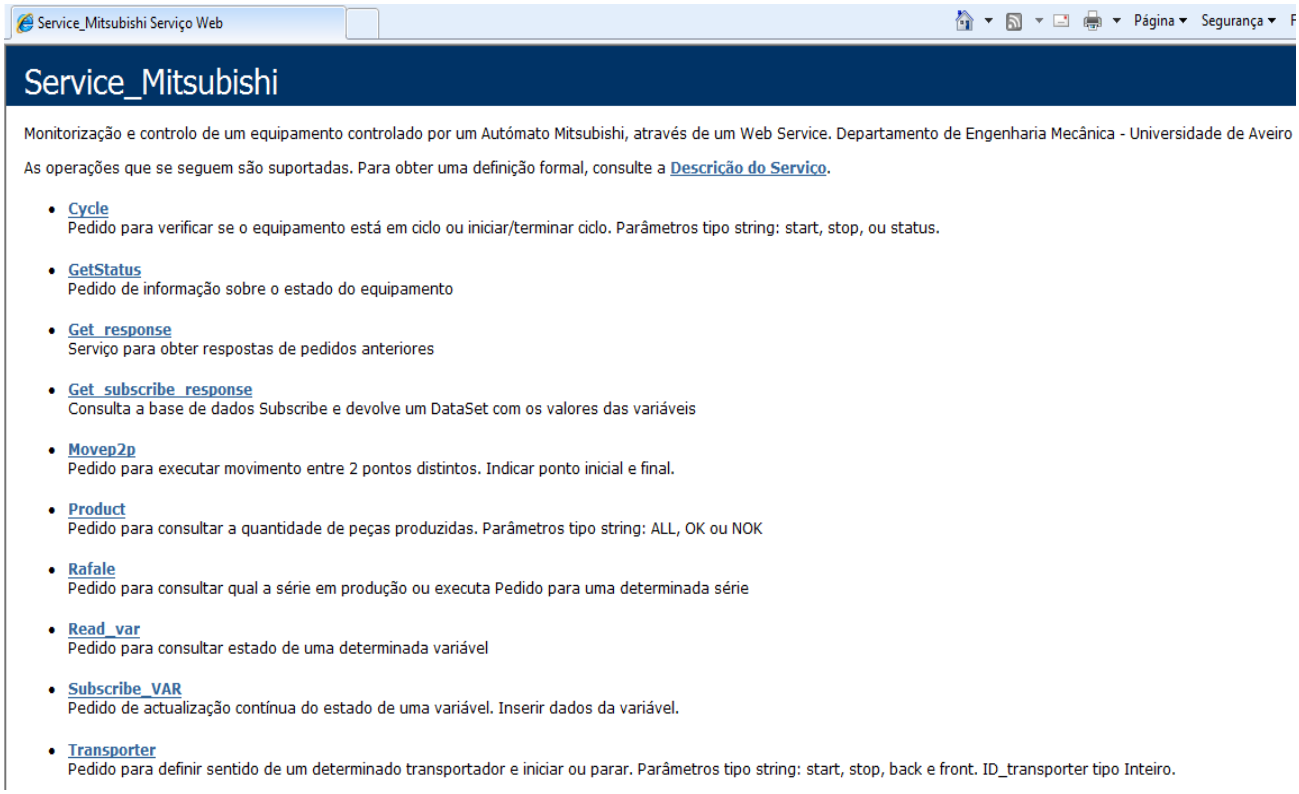


Figura 30 - Acesso a um Web service, através do *Browser*, onde é possível ver alguns dos serviços disponíveis.

O consumo de um serviço no browser é também simples, como se pode ver na Figura 31, sendo que após seleccionar o serviço o cliente deve introduzir os dados necessários e caso o serviço venha a necessitar de dados de entrada, ”clica-se ” no botão “Invocar” para executar o serviço.

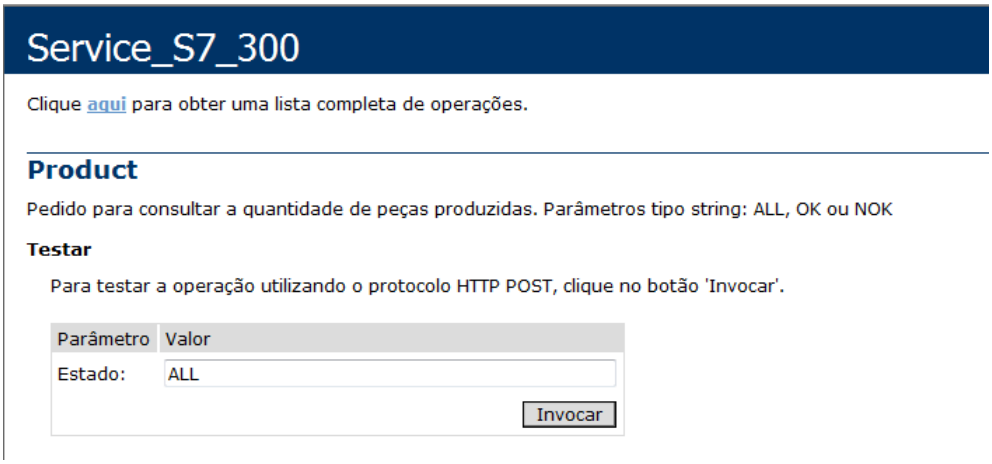


Figura 31 - Consumo do serviço *Product* no *browser.*

Após executar o serviço o cliente receberá uma resposta idêntica à representada na Figura 32, indicando se o serviço foi executado com o sucesso (OK), o estado do serviço anterior (3 – o serviço anterior tinha sido concluído com sucesso) e o estado do equipamento (6 – o equipamentos está disponível, em modo manual ou remoto).

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">OK;3;6</string>
```

Figura 32 - Resposta à invocação do serviço *Product.*

A integração da monitorização do equipamento, o campo *Rstate*, da tabela de troca de informação (com), pode criar inicialmente alguma confusão ao cliente, tendo em conta que dependendo do estado do equipamento será possível realizar determinados serviços/acções. Se o equipamento estiver em modo de ciclo automático (5), ou em defeito (7), os clientes apenas poderão fazer monitorização ou alterar séries de produção do equipamento. Caso o equipamento esteja em modo manual ou remoto (6), está disponível para todo o tipo de acções. O último parâmetro é o de falha na ligação (8), o que, como é óbvio, não permite a realização de qualquer pedido.

O modelo de interacção cliente-servidor pretendido para os Web services é ilustrado no diagrama da Figura 33.

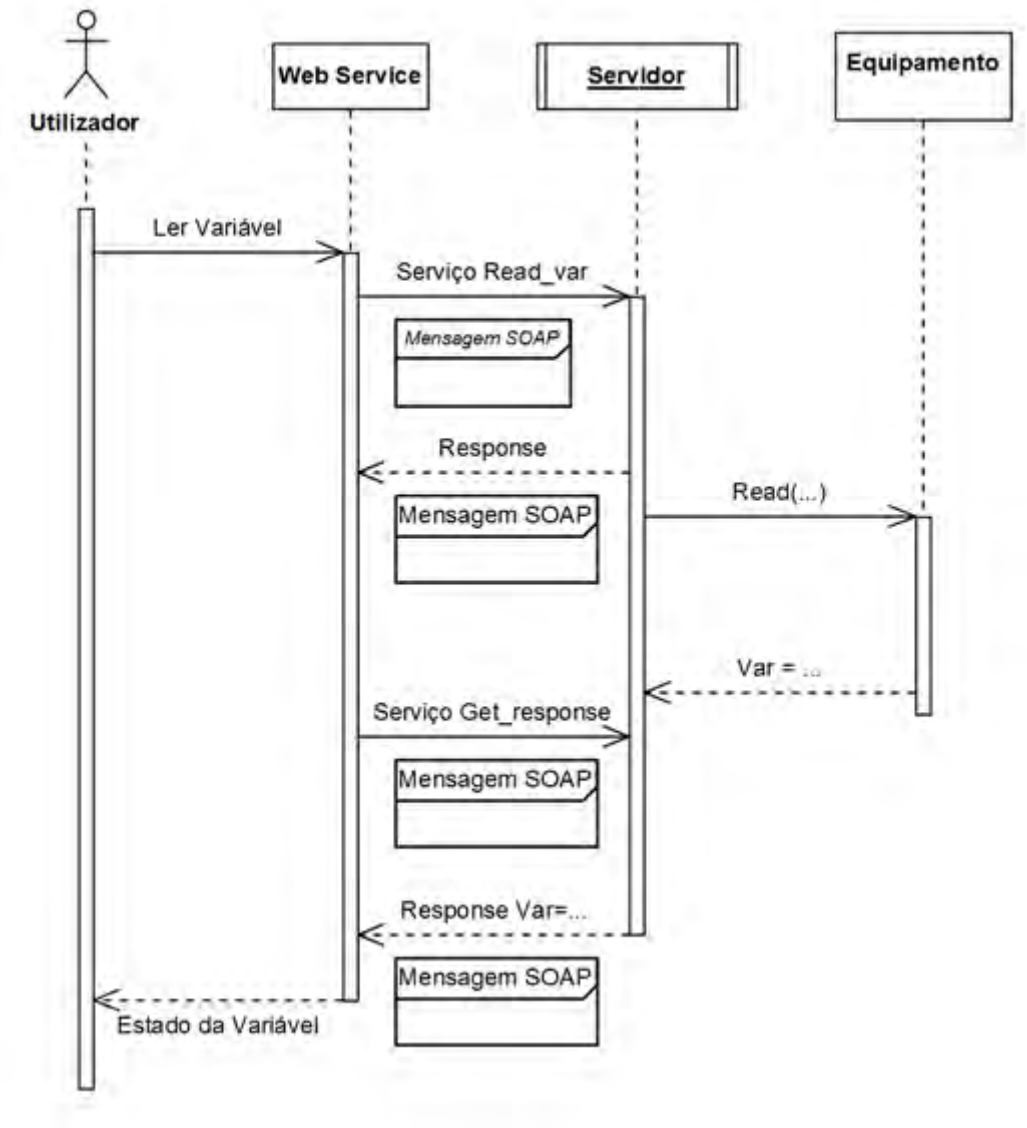


Figura 33 - Diagrama de interacção para um serviço tipo.

Como o diagrama exemplifica, a interacção entre aplicações inicia-se sempre com uma necessidade do utilizador/cliente. O cliente invoca o serviço pretendido e obtém uma resposta relativamente à entrada ou não do pedido na base de dados, no servidor.

O pedido entra na base de dados e é processado pela aplicação local, que comunica com o equipamento e obtém a devida resposta, colocando-a na base de dados. O acesso do cliente aos dados pretendidos, implica que seja solicitado o serviço *Get_response* para obter os dados.

Naturalmente, para o bom funcionamento dos serviços propostos, será necessário preparar a programação dos equipamentos para receber pedidos e ordens remotas, o que se reflecte na criação de toda uma estrutura para permitir o desenvolvimento deste tipo de arquitecturas.

4.2.4 Interacção entre aplicações

A comunicação entre aplicação cliente (*Web services*) e aplicação local organizada, estruturada e sem interferências entre pedidos de aplicações cliente, impôs que fosse delineado um campo para estabelecer esta troca de dados entre as aplicações.

As tabelas, da base de dados, para a troca de dados e informações entre aplicação cliente e servidora, denominadas por "com", têm a coluna *"Status"*, na qual é colocado o índice correspondente a um estado da aplicação local e/ou recurso (Figura 34). O índice deve ser actualizado tanto pela aplicação cliente como pela aplicação local, sendo os índices a utilizar pelas aplicações cliente e servidor:

- 0 – *Start* (Início): iniciar pedido, o programa de controlo local ao ler o valor 0 verificará os restantes dados para tomar conhecimento das ordens ou pedidos.
- 1 – *Busy* (Ocupado): o recurso está a processar uma ordem ou iniciar.
- 2 – *Error* (Erro): ordem não concluída ou o recurso não está em serviço.
- 3 – *Completed* (Concluída): ordem concluída com sucesso pelo recurso.

Este campo permite a troca de informação, sem incoerências, entre aplicações, permitindo a implementação do modelo Cliente/Servidor. Desta forma, a aplicação cliente, ao efectuar um pedido, irá alterar o estado do campo status para o valor **0**, o que significa que a aplicação local deverá iniciar uma tarefa e alterar o campo *Status* para **1** *(Busy)*. A conclusão da tarefa leva a aplicação local a indicar o resultado da tarefa executada: **3 -** caso a conclua com sucesso e, se necessário, o cliente poderá realizar um pedido para obter uma resposta, ou **2** - se por algum motivo não pode realizar a tarefa ou ocorre algum erro.

ID	Ordem	Var1	Var2	Var3	Value	Status
1	read	X	1		2	0

Figura 34: Exemplo de tabela de troca de informação entre aplicação cliente e servidora.

Apenas o campo *Status* não é suficiente para que seja possível implementar um sistema de supervisão e controlo através de *Web services*, pois este campo apenas reflecte o estado da aplicação local e dos pedidos efectuados pela aplicação cliente.

Para além da coluna Status foi criada uma coluna (R_state) para identificar o estado dos processos ou equipamentos (Figura 35), para indicar se estão em produção/ciclo automático, disponíveis/modo Manual

ou remoto, com alarmes ou defeitos ou se a ligação à aplicação local não está efectuada Para colmatar esta questão foram usados índices diferentes dos anteriores para não gerar confusões, sendo os seguintes:

- 5 – Em produção ou ciclo automático: não poderão ser efectuadas ordens que interfiram com os processos.
- 6 – Disponível: o recurso ou equipamento está disponível para todo o tipo de ordens ou pedidos, poderá estar em modo manual ou simplesmente parado.
- 7 – Defeitos ou alarmes: o recurso está com defeitos, alarmes ou emergências, estando bloqueado para todo o tipo de ordens de acção.
- 8 – Defeito na ligação: é impossível comunicar com o recurso/equipamento.

ID	Ordem	Var1	Var2	Var3	Value	Status	R_state	Hora	Data
1	read	MB	2	Byte		0	6	12:45:36	08-05-2009

Figura 35 - Estrutura final das tabelas de troca de dados entre aplicações.

O uso destes índices é possível se o código do *Web service* obedecer à seguinte estrutura de programação:

```
Public Function test (ByVal Point1 As String, ByVal Point2 As String) As String
    'Lê Base de dados
    If Status=3 (Completed) or Status=2(Error) Then
        If R_state=5 (em ciclo) or R_state=7(em defeito) Then
            'Escreve pedido na base de dados, nesta situação apenas
poderá fazer pedidos para realizar monitorização do recurso
            Return "OK;Satus;R_state" ou "NOK;Satus;R_state" Caso seja
para uma ordem para executar uma tarefa
        Else if R_state=6 Then
            'Escreve pedido ou ordem na base de dados
            Return "OK;Satus;R_state"
        Else
            Return "NOK;Satus;R_state"
        End If
    Else
```

```
            Return "NOK;Satus;R_state"
        End If
End Function
```

O serviço começa por verificar o estado do recurso (coluna *Status*) e caso este tenha o estado *Completed (3)* ou o *Error (2),* poderá enviar pedidos ou ordens, embora primeiro verifique o estado do recurso ou equipamento. Se o estado do recurso ou equipamento estiver em ciclo ou produção (estado 5), ou em defeito (7), poderá apenas enviar pedidos para monitorização, no entanto, se estiver disponível (6) poderá enviar todo o tipo de pedidos ou ordens, e caso a ligação não esteja estabelecida (8), não permite qualquer envio.

Portanto, um *Web service*, após verificar os estados na tabela de troca de dados ("*com*"), actualiza a tabela com os dados relativos ao pedido pretendido, para que o programa de controlo local identifique a ordem ou pedido a efectuar, e inicie a tarefa. O *Web service* tem que actualizar a coluna *Status* com o índice '0', ou seja, *start*, para que o programa de controlo local dê inicio à ordem ou pedido a executar.

O serviço devolve "OK" caso a actualização da tabela seja bem sucedida, ou "NOK" nos seguintes casos:

- Caso estejam a ser processadas tarefas, modo *Busy (1, Status)* ou *Start (0*);
- Caso o equipamento não possa executar uma ordem: equipamento em ciclo (5) ou defeito (7) (*Rstate*);
- A ligação não esteja estabelecida entre aplicação local e o equipamento (8, *Rstate*);
- Se não conseguir actualizar a tabela, devido a algum problema de comunicação com a *SGBD*.

Para além de devolver o resultado "OK" ou "NOK", os *Web services* retornam também os estados do processamento dos pedidos ou ordens pela aplicação local (Coluna Status) e o estado dos equipamentos (Coluna R_state), para que a aplicação cliente possa saber os estados.

A aplicação local também terá uma estrutura baseada neste conceito para efectuar a leitura da tabela de comunicação e realizar as tarefas, possibilitando que o recurso a este método seja produtivo e com resultados positivos.

Segue-se um exemplo da estrutura de programação da aplicação local, para receber e interpretar os dados da aplicação cliente e efectuar as ordens ou pedidos, colocando posteriormente, na tabela respectiva à troca de dados entre aplicações e na tabela de respostas, o resultado e resposta, respectivamente, do pedido ou ordem efectuado pela aplicação cliente.

```
Private Sub Timer_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer.Tick
    Leitura da base de dados (tabela de troca de dados)
```

```
If state = 0 Then 'Se o estado for 0 - start

        Actualiza o estado na base de dados para “Busy” - indica que está em
execução um pedido/ordem (state = 1)

        Verifica qual a ordem ou pedido

        If dados/parâmetros OK Then

                Executa as ordens ou pedidos

                Verificação da comunicação

                Actualiza a base de dados com o resultado da tentativa de
execução do pedido/ordem (completed - 3 ou em caso de erro -                2)

                Actualiza a base de dados com o resultados caso seja
necessário

        Else

                Actualiza a base de dados com o estado 2 - erro

        End If

    End If

End Sub
```

O estado do equipamento ou recurso, na coluna *R_state* da tabela de troca de dados, terá de ser constantemente actualizado pela aplicação local, para que o sistema funcione de forma correcta.

Deste modo, será possível interpretar todos os pedidos ou ordens efectuados pela aplicação cliente, como também não se pode sobrepor um pedido/ordem a um que já esteja a ser processado pela aplicação local. A aplicação cliente pode ainda saber se o seu pedido/ordem foi executado com sucesso ou não, e no caso de haver retorno de valores pode fazer um “*pull*” desses valores acedendo à base de dados.

4.3 Exemplos de aplicação

O capítulo apresenta a integração dos *Web services* propostos numa aplicação cliente.

A demonstração dos serviços propostos e sua integração será feita utilizando dois recursos - um pequeno tapete transportador com um moto-redutor e duas foto-células aplicadas no início e fim do tapete, sendo o tapete controlado por um autómato *Siemens* S7-314; uma linha de transferência de paletes, composta por seis barragens, seis paletes e respectivos detectores, o recurso é controlado por um autómato *Mitsubishi* A1NS.

A integração dos *Web services* numa aplicação cliente, desenvolvida em *Visual Studio 2005 .Net*. decorre em várias etapas.

O primeiro passo para a inclusão de um *Web service* numa aplicação é a selecção da opção "*add Web Reference*", no menu "*Project*" (Figura 36), à qual se segue a abertura de uma caixa de diálogo para selecção do serviço, conforme a Figura 37.

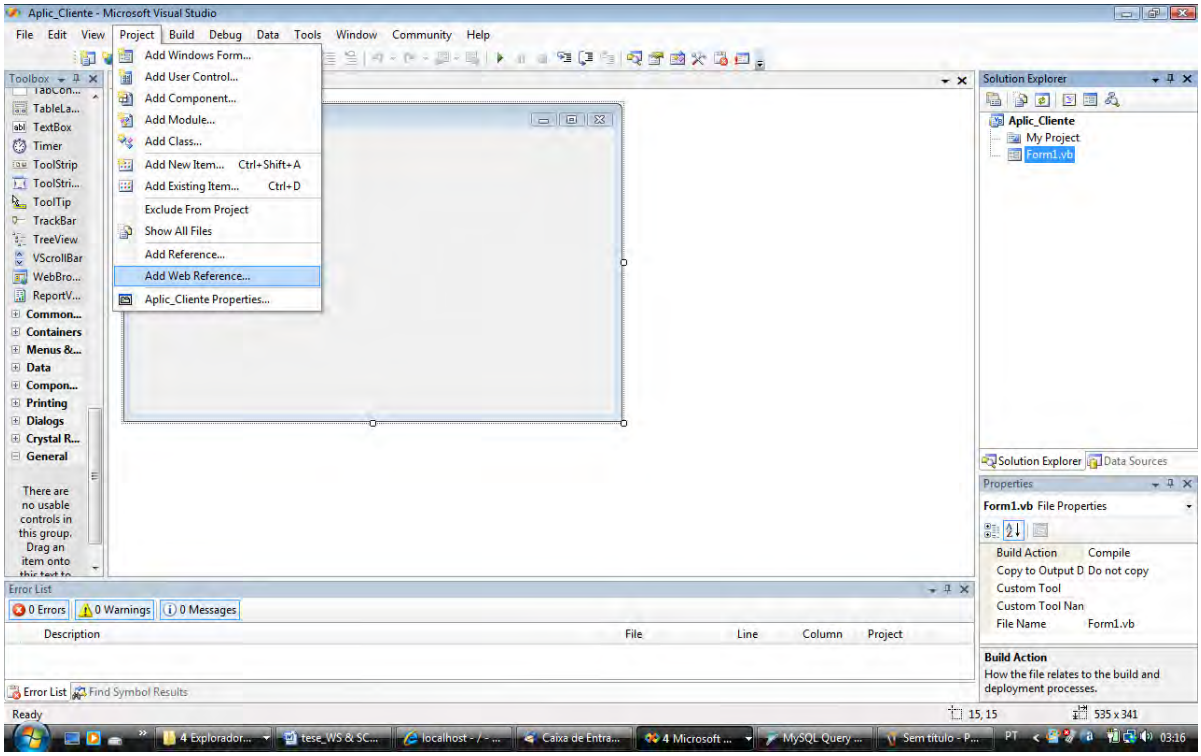


Figura 36 - Selecção da *opção* para adicionar uma referência *Web.*

Na caixa de diálogo deve-se inscrever o *URL* do serviço pretendido, clicando de seguida no botão "*Go*", para aceder ao *Web service* proposto e ver a descrição do serviço (Figura 38).

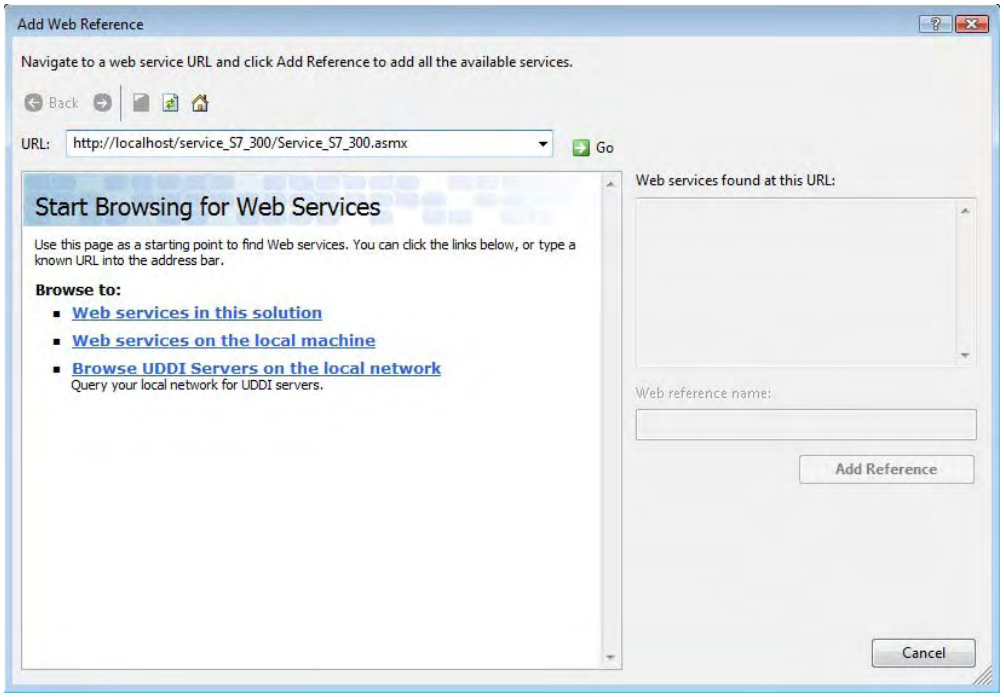


Figura 37 - Caixa de diálogo de pesquisa e selecção do URL do *Web services*pretendido.

O último passo para concluir a integração do serviço na aplicação passa por atribuir um nome ao serviço, "*Web reference name*", e "clicar" no botão "*Add Reference*" para concluir o processo.

Na Figura 38 podemos ver a integração do *Web service* proposto para o *PLC* da *Siemens S7 300*, que será denominado "tapete". O referido *PLC* está associado ao tapete transportador, conforme mostra a Figura 39, onde se representa a integração do *Web service* proposto para o *PLC* da *Mitsubishi*, que está associado à linha de transferência de paletes, sendo desta forma apelidado de "linha".

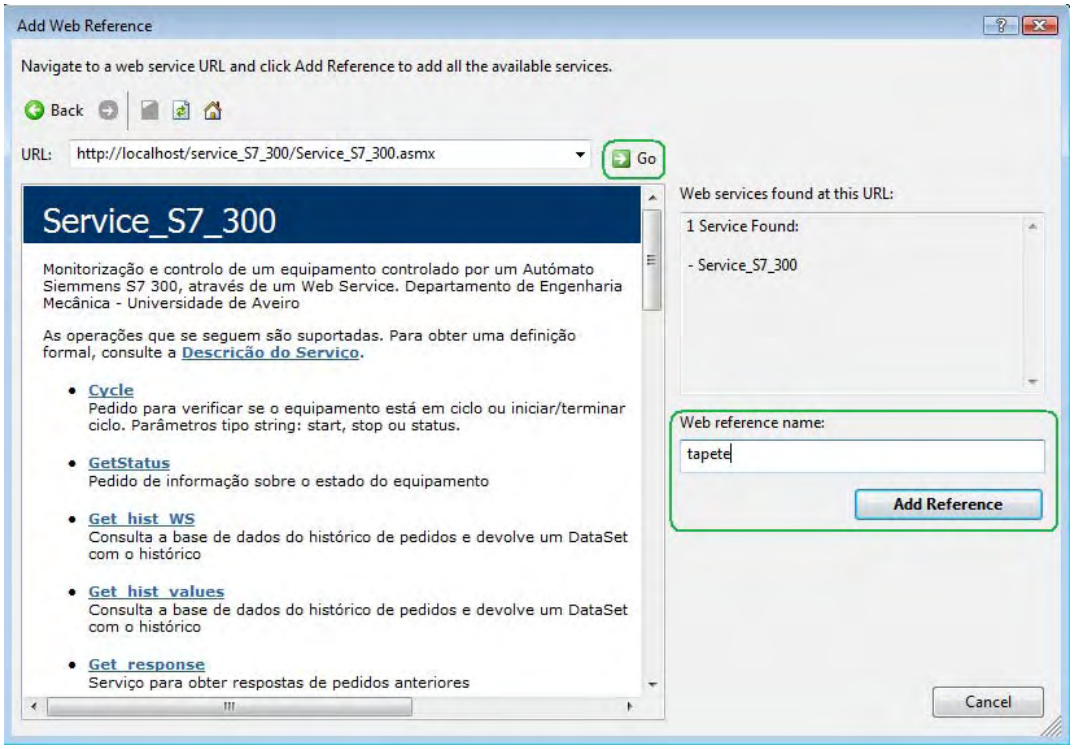


Figura 38 - Validação e nomenculatura dada ao *Web service* para o tapete controlado para pelo *PLC Siemens.*

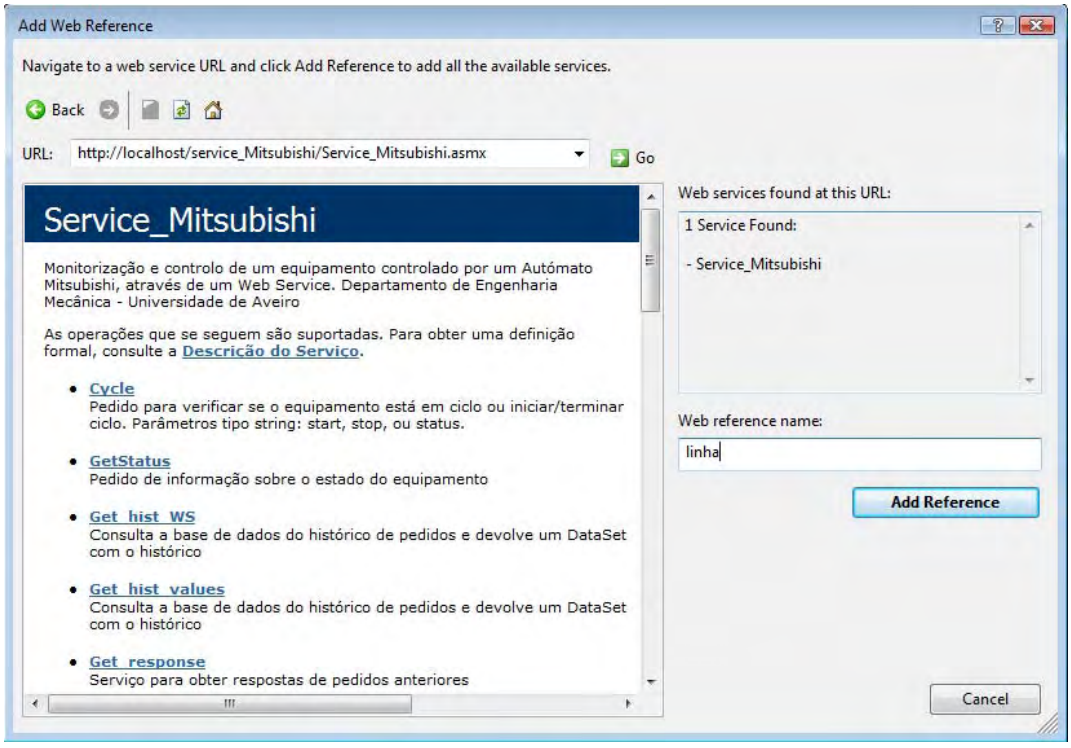


Figura 39 - Validação e nomenculatura dada ao *Web services* para a linha de transferência controlada pelo *PLC Mitsubishi*.

Adicionados os serviços propostos à aplicação, podemos conferir que estão presentes na árvore do "*Solution Explorer*" da aplicação (Figura 40).

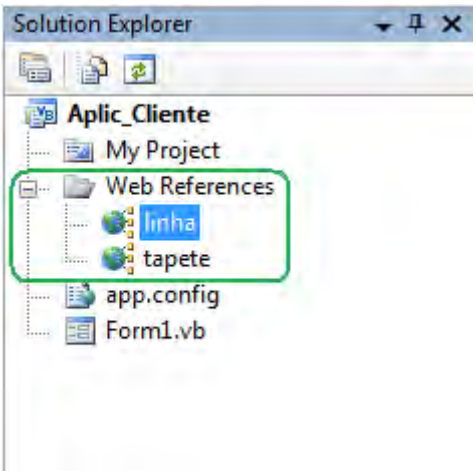


Figura 40 - *Solution Explorer* da aplicação cliente com as referências *Web* adicionadas.

Para poder utilizar todos os serviços disponíveis será necessário criar as respectivas classes no programa, como indicado na Figura 41.

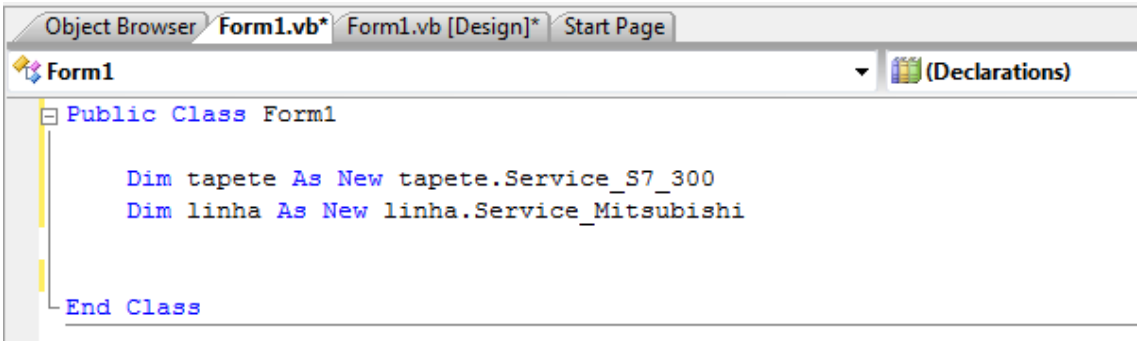


Figura 41 - Definição e criação das classes associadas aos *Web services.*

Definidas as classes respectivas aos *Web serices* propostos para os *PLC's Siemens S7 300 e Mitsubishi,* resta que sejam utilizados todos os serviços, de acordo com a Figura 42 que exemplifica a integração do serviço "*GetStatus*" para determinar o estado do equipamento e da aplicação local.

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        estado.Text = tapete.G
    End Sub
End Class
```

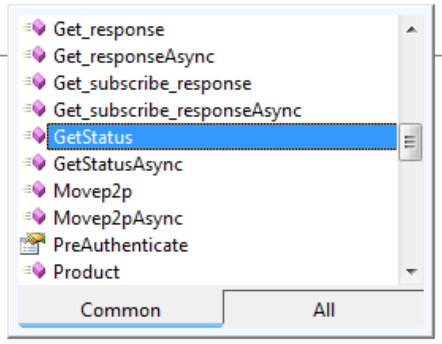


Figura 42 - Exemplo de utilização das funções disponíveis.

Com os serviços propostos adicionados, foi desenvolvida uma pequena aplicação cliente que permite monitorizar e controlar o tapete e a linha de transferência de paletes. A Figura 43 representa a interface da aplicação.

Figura 43 - Interface da Aplicação Cliente de demonstração.

O exemplo de aplicação permite comprovar a funcionalidade do sistema criado, e apesar dos recursos não se tratarem de aplicações industriais e de serem simples, pode-se confirmar a facilidade e utilidade de integração dos serviços criados, reflectindo-se na funcionalidade e sem falhas, sendo quase instantâneos.

Outros exemplos de aplicação

Robot ASEA– Aveiro Norte, Atena, Utilzás

Neste momento está em estudo a adaptação de uma aplicação de controlo de um robot ASEA, para trabalhar em modo remoto, como cliente *Web.* A aplicação em questão está a ser desenvolvida pelo aluno Sérgio Paiva da Escola de formação profissional Aveiro Norte, tendo uma interface simples onde é possível controlar os vários eixos do robot ou definir as coordenadas cartesianas do *robot*.

Esta aplicação serve-se do *MX Component* da Mitsubishi para comunicar com o autómato da *Mitsubishi* que funciona como controlador do *robot*.

Localmente pretende-se substituir esta aplicação, pela aplicação local desenvolvida no decurso desta dissertação e integrar os *Web services* na aplicação de controlo do *robot* permitindo que esta funcione através da *Web*, como cliente,

A figura seguinte ilustra a interface da aplicação.

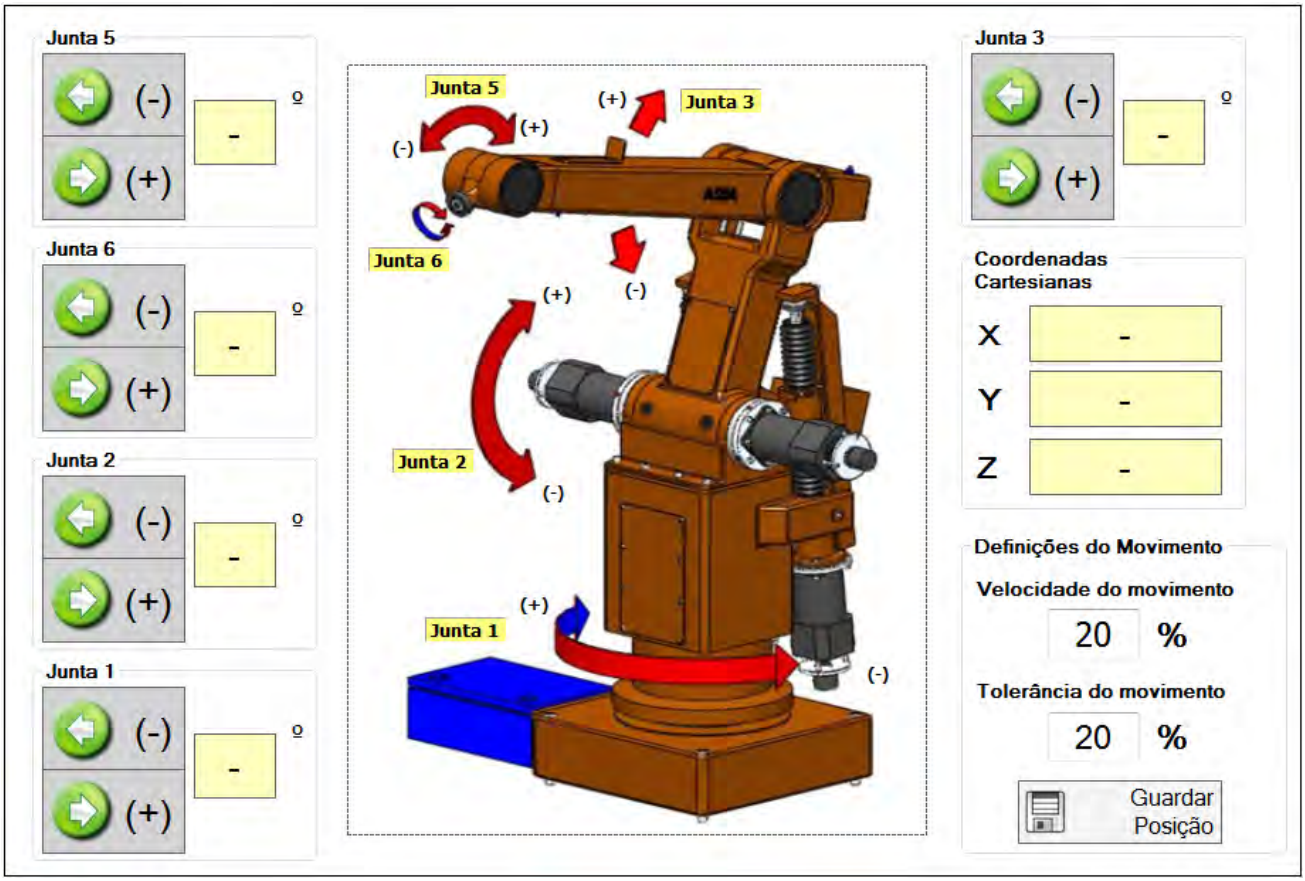


Figura 44 - Interface de monitorização e controlo do robot ASEA.

Perfiladora – Empresa Portaveiro

Esta máquina (Figura 45) produz perfis de chapa, a partir de bobines de chapa, para montagem de portões automáticos articulados e permite a criação de perfis distintos e com vários comprimentos. A perfiladora é controlada por um autómato da *OMRON* e dispõe de uma *HMI* para escolha e edição dos parâmetros de produção.

Figura 45 - Perfiladora da empresa Portaveiro.

Trata-se de um bom exemplo onde será possível a integração do sistema desenvolvido criando uma aplicação cliente que permitiria a escolha do perfil a produzir, o comprimento do perfil e a quantidade peças a produzir.

5 Resultados e análises qualitativas

Neste capítulo pretende-se analisar a qualidade, a funcionalidade e a acessibilidade do sistema criado e dos serviços, para além de avaliar o resultado da sua integração numa aplicação cliente.

De lembrar que o objectivo deste trabalho é a implementação de um sistema que permita a criação de pequenas aplicações *SCADA* via *Web*, ou seja, permitir que um programador possa criar uma aplicação *SCADA* remota para um ou mais equipamentos. Outro pressuposto é a análise dos tempos de processamento dos dados de toda a arquitectura, desde o pedido pelo cliente até ao momento em que este consegue obter o resultado do seu pedido.

Analisar as funcionalidades e características da solução proposta e compará-las com servidores *OPC* e mesmo com *softwares SCADA* incorpora também uma das metas do trabalho.

A arquitectura implementada mostrou-se bastante eficaz face ao objectivo pretendido, o que significa que o desenvolvimento de um sistema que possa funcionar como um *Middleware* de interacção entre aplicações remotas e equipamentos industriais, permitindo a monitorização e controlo dos equipamentos. A eficácia mantem-se, mesmo com mais do que um equipamento ligado ao servidor e tempos de resposta relativamente curtos e capaz de disponibilizar todas as informações e dados pretendidos.

Os protocolos e as rotinas de comunicação implementados são eficazes, com tempos de resposta bons e permitem o acesso a todo o tipo de variáveis e informações dos equipamentos, disponibilizando funções semelhantes às existentes em *softwares SCADA* e servidores *OPC*, contribuindo de forma favorável para o bom funcionamento de todo o sistema, no entanto, com a vantagem de ser uma solução *freeware*, permitir a interoperabilidade entre sistemas e aplicações e a possibilidade de adaptação fácil para diferentes equipamentos.

Os vários serviços criados, para determinados tipos de acções, mostram-se flexíveis para o mais variado tipo de equipamentos e processos, sendo independentes do tipo de equipamento pelo facto de que os únicos parâmetros que mudam são as variáveis a que a aplicação local deve aceder ou alterar no equipamento.

A integração dos serviços em aplicações é bastante simples (capítulo 4.3), tal como utilizar os serviços através de um *browser*. Da mesma forma que são de fácil integração, também é simples tirar partido das potencialidades de cada serviço, uma vez que tanto os serviços como os parâmetros de entrada, em grande parte, são intuitivos. No entanto, em alguns casos poderá haver problemas, quanto aos parâmetros de entrada, com os serviços a não estarem preparados para limitar as opções ou mesmo para interpretar a diferença entre maiúsculas e minúsculas, o que pode levar a alguns erros iniciais por parte dos clientes.

A tabela 9 apresenta uma comparação entre o sistema criado, denominado por WSCD (*Web services* para sistemas *SCADA*), e um servidor *OPC*, o tipo de aplicação mais próxima do sistema desenvolvido. Em avaliação estarão as características e valências do sistema criado em função de servidores *OPC*.

A comparação com os servidores *OPC* é importante não só pela proximidade entre os sistemas, mas também pelo facto dos servidores *OPC* se centrarem nas especificações da *OPC Foundation.* O sistema criado tenta seguir as especificações *OPC*, com especial foco sobre a especificação *OPC XML DA*, que é a que rege os modelos de criação de *Web services* para a troca de dados e informações com equipamentos.

Segue-se a tabela comparativa das qualidades de cada sistema:

Tabela 9 - comparação entre caracteríticas dos sistema desenvolvido e servidores *OPC*.

	Sistema desenvolvido WSCD	**Servidores *OPC***
Instalação	No servidor local, a instalação será simples.	Instalação simples e clara.
Interface	Interface simples e intuitiva com poucos grafismos.	Na sua maioria a interface é simples e intuitiva, sendo que alguns já contemplam interfaces gráficas para criação de projectos de supervisão e controlo.
Drivers e bibliotecas dinâmicas	Até ao momento contém bibliotecas e drivers para *Mitsubishi*, *Siemens*, *Omron* e Fagor, não estando preparado para todas as séries de cada marca. Contudo, os protocolos utilizados, à excepção da *Mitsubishi*, são implementados ou soluções conjuntas com bibliotecas dinâmicas de projectos *freeware*.	Dependendo de cada servidor e tendo em conta se são ou não proprietários, mas as soluções mais flexíveis, contemplam drivers para as mais variedades de dispositivos, não usando protocolos proprietários.
Conectividade	Os serviços criados estão disponíveis através da *Web*, para serem integrados em aplicações remotas.	A maioria não suporte soluções remotas, estando integradas, servindo como um *Middleware*. Actualmente, alguns dos servidores também já têm soluções para permitir a integração em aplicações remotas via *Web.*
Conectividade com equipamentos	Actualmente, disponível com a porta série, mas é possível adaptar para E*thernet.*	Algumas soluções permitem, para além da comunicação série, *Ethernet* e *Modbus*, ou outras soluções proprietárias.
Especificações *OPC*	Implementado baseando-se em muitos aspectos nas especificações *OPC*, principalmente a *OPC XML* DA.	Todos são baseados nas especificações *OPC.*

Diagnóstico de protocolos	Diagnóstico de erros na comunicação com os equipamentos apenas disponível para alguns protocolos, na aplicação local.	Suportam ferramentas de diagnóstico avançado.
Programas de teste ou simulações	Disponibiliza uma interface local que permite de forma muito simples realizar acções idênticas às dos *Web services*, no entanto, não suporta a criação de programas de teste ou simulação.	Alguns dos servidores *OPC* permitem a simulação de projectos, ou mesmo a criação de projectos para supervisão e controlo.
Comunicações em simultâneo	Permite a utilização por vários clientes remotos, sendo que por equipamento, não aceita pedidos simultâneos, são aceites e processados separadamente.	Permitem clientes simultâneos.
Detecção de endereços dos dispositivos	Quando há necessidade, os endereços dos dispositivos são configurados na aplicação local.	Alguns dos servidores *OPC*, principalmente, as soluções proprietárias.
Arquitectura	Arquitectura cliente-servidor descentralizada e orientada a objectos.	Arquitectura cliente-servidor descentralizada e orientada a objectos.
Interoperabilidade	Os serviços *Web*, devido aos protocolos em que se baseia, são completamente interoperáveis, nem dependendo da plataforma de desenvolvimento. A aplicação local é uma aplicação desenvolvida em *Visual Studio*, sendo uma aplicação tipo "*Windows*".	Algumas soluções permitem a interoperabilidade entre sistemas.
Suporte para componentes activex ou outros	Actualmente não contempla, mas está em estudo, a hipótese de considerar a integração de servidores *OPC* ou componentes *activeX*, para colmatar a falta de protocolos.	Algumas aplicações não proprietárias já têm integrado a possibilidade de integração de controlos *activeX*.
Leitura e escrita	Permite o acesso a todo o tipo áreas de memória dos equipamentos.	Permite o acesso a todo o tipo áreas de memória dos equipamentos.
Gestão de alarmes e defeitos	Não tem gestão de alarmes e defeitos, no entanto, pode pedir a actualização contínua de variáveis associadas a alarmes e defeitos. Contempla também um serviço que permite saber o estado dos equipamentos.	Algumas das soluções existentes permitem uma total gestão de alarmes e defeitos.
Históricos	Permite a consulta de históricos.	Permite a criação de históricos de processos.

Ferramentas apropriadas para determinadas acções	Alguns dos serviços criados são determinados a acções específicas, sendo devidamente processados na aplicação local. Serviços reutilizáveis entre aplicações e facilmente integrados e consumidos.	Não usa ferramentas específicas, sendo que apenas são criadas *tags* associadas a variáveis para determinadas acções.
Monitorização do estado dos equipamentos, habilitando ou não determinadas acções	Aplicação local é configurada de maneira a que seja feita a monitorização do estado dos equipamentos, impedindo os clientes de executar pedidos que possam interferir com o funcionamento dos equipamentos.	Não têm este tipo de controlo, mas ao criar *tags* podem-se definir apenas de leitura, para não interferir com o funcionamento dos equipamentos.
Actualização contínua de variáveis	Um dos serviços criados permite inserir variáveis para actualização contínua na base de dados, sendo o seu estado constantemente actualizado. O cliente apenas terá que fazer um "*pull*" dos dados, um conceito inteiramente baseado na especificação *OPC XML DA*.	Não suportam este conceito, mas o acesso ao estado das variáveis é algo semelhante. As variáveis são indexadas a *tags* (itens), que por sua vez estão organizados em grupos, bastando aceder às *tags* criadas para saber o estado das variáveis.

Como se pode constatar, após analisar a tabela anterior, o sistema desenvolvido e os *Web services* assemelham-se aos servidores *OPC*, na medida em que disponibilizam ferramentas aos clientes semelhantes às de um servidor *OPC*, tirando partido de ser um sistema distribuído que habilita clientes remotos, a criar aplicações do tipo *SCADA* através da *Web*.

A arquitectura desenvolvida não se pode comparar a um *software SCADA*, uma vez que este é toda uma ferramenta que permite a monitorização e controlo de um equipamento. Com a arquitectura desenvolvida habilitam-se os clientes a criar as suas próprias aplicações *SCADA*, sendo que se tentou disponibilizar um conjunto de serviços viáveis que forneçam as mesmas possibilidades de um *software SCADA*:

- Acesso a todo o tipo de variáveis;
- Consultar os estados dos recursos;
- Consulta de históricos;
- Comunicação independente com os equipamentos;
- Acesso aos estados e valores das variáveis o mais próximo possível de em tempo real.

Outro ponto sobre o qual é importante uma análise é o tempo de resposta dos serviços, o mesmo que dizer tempo gasto desde que uma aplicação cliente executa um pedido até obter uma resposta ao

pedido. Neste sentido foram criadas rotinas na aplicação cliente apresentada no ponto 4.3 para determinar o tempo de alguns serviços. Os serviços utilizados para analisar o tempo de resposta foram os serviços que necessitam executar o serviço "*Get_response*" para obter os dados, isto porque são as situações que englobam mais transferências de dados e comunicações.

No gráfico representado na Figura 46 pode-se ver o tempo de resposta para cerca de 100 pedidos consecutivos, em que para esta série de pedidos a aplicação cliente estava a correr na mesma rede que o que a aplicação local, o que pode influenciar de forma positiva os tempos obtidos.

Neste caso a aplicação local consulta a base de dados para verificar a existência de novos pedidos em intervalos de 500ms, ou seja, a cada meio segundo a aplicação local verifica se foi introduzido um novo pedido na base de dados e processa-o.

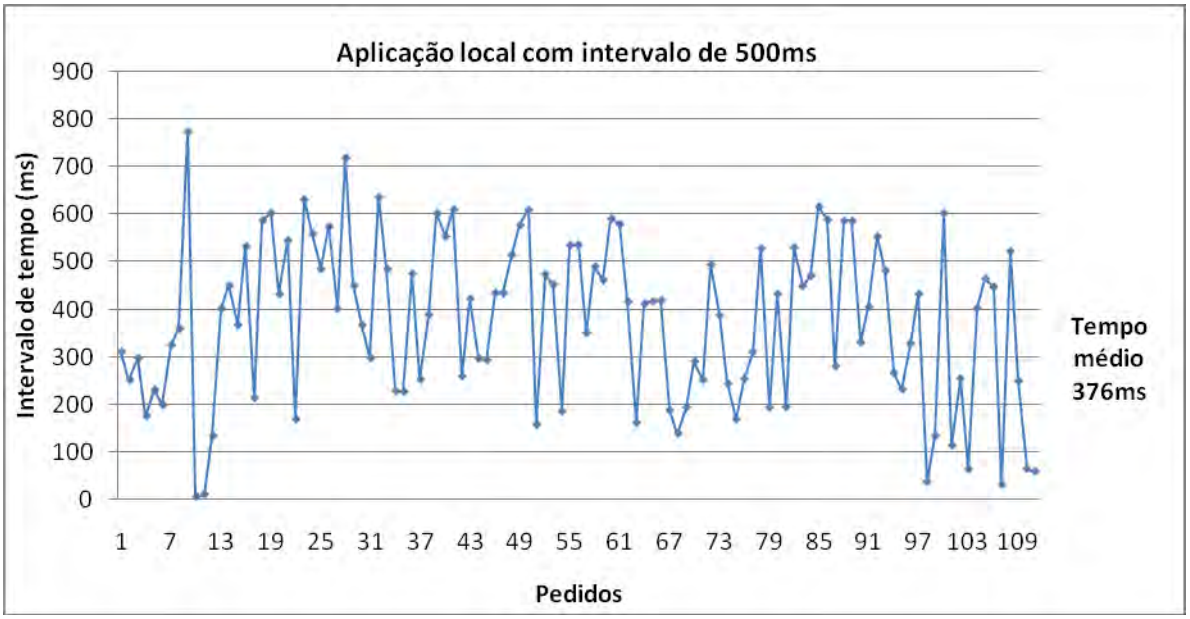


Figura 46 - Gráfico do intervalo de tempo de execução de um pedido, com aplicação cliente com intervalo de 500ms.

Como se pode verificar o tempo médio de execução de um serviço é de 376 milissegundos, o que representa um tempo reduzido, agregando vários passos:

1. Leitura e escrita na base de dados pela aplicação cliente;
2. Leitura da base de dados pela aplicação local;
3. Processamento dos dados e comunicação com o equipamento;
4. Escrita na base de dados com os resultados, pela aplicação cliente;
5. Leitura dos resultados na base de dados, pela aplicação cliente.

No gráfico podemos verificar que há intervalos de tempo muito curtos, justificados pelo facto de não estar contemplado, na rotina executada, a espera para envio do pedido se a aplicação local estiver ocupada,

ou seja, sempre que era enviado um pedido e a aplicação estava ocupada, o serviço terminava dando informação de que a aplicação local estava ocupada. Desta forma, é apenas executado um passo, o de leitura da base de dados pela aplicação cliente.

Pelo contrário, existem alguns pedidos que têm um intervalo maior cerca de 800ms, consequência do intervalo de tempo com que a aplicação local verifica a base de dados, bem como as notórias diferenças de tempo entre aplicações.

Para que se possa analisar a influência do intervalo de tempo com que a aplicação de tempo efectua a leitura da base de dados realizaram-se ensaios com outros intervalos de tempo, conforme o gráfico para um intervalo de 100ms na aplicação local (Figura 47).

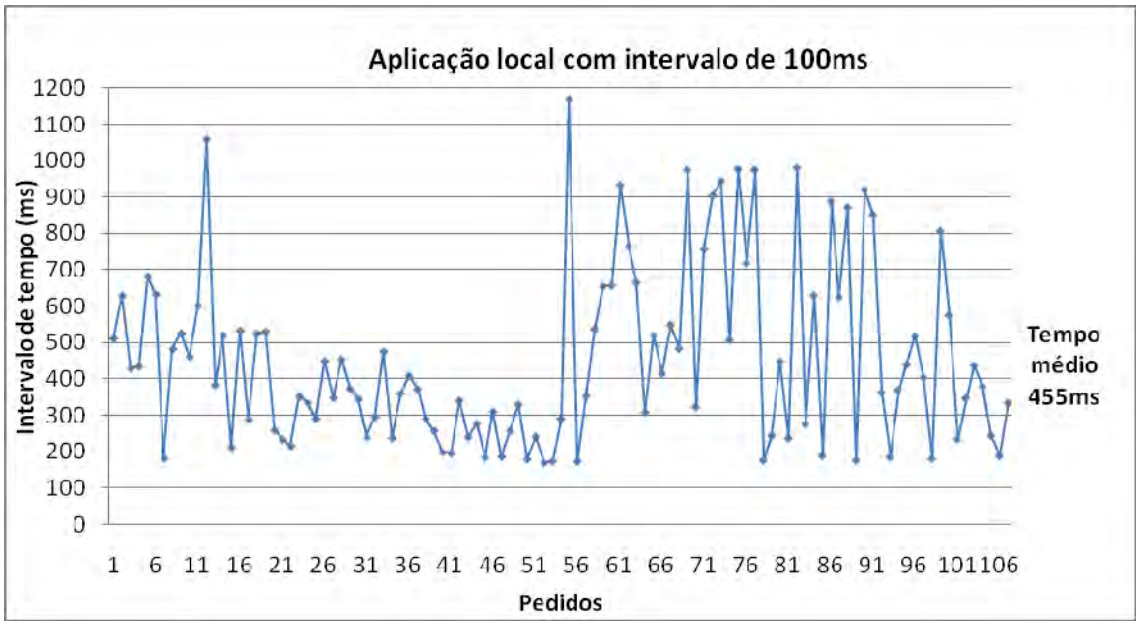


Figura 47 - Gráfico do intervalo de tempo de execução de um pedido, com aplicação cliente com intervalo de 100ms.

Ao diminuir o intervalo de leitura da aplicação local, em muitos dos pedidos o tempo total de execução fica compreendido entre valores de 180ms e os 400ms, o mesmo que tempos um pouco inferiores à situação anterior No entanto, ao analisar todos os valores, foi possível confirmar que existem outros valores muito elevados e uma variação significativa, o que leva a crer que o tempo de 100ms é substancialmente curto e provoca alguma instabilidade na aplicação local.

Nos tempos analisados, nas duas situações anteriores, existe uma grande variação entre os tempos dos pedidos. Na Figura 48 mostra-se a análise do tempo de execução dos pedidos para um intervalo de leitura da base de dados, pela aplicação cliente, 200ms.

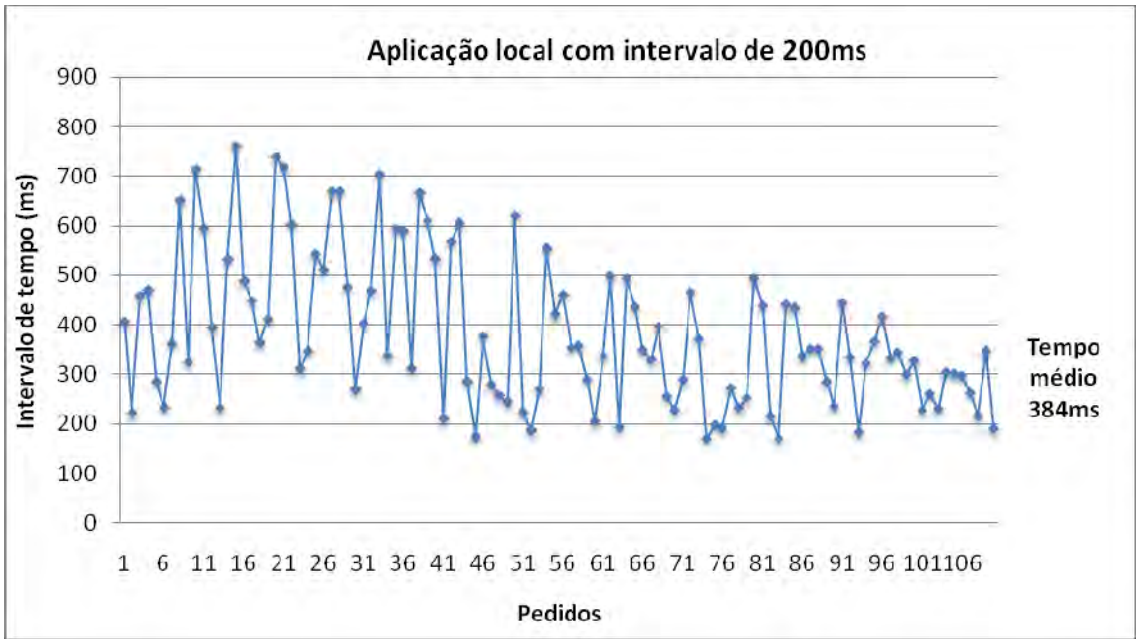


Figura 48 - Gráfico do intervalo de tempo de execução de um pedido, com aplicação cliente com intervalo de 200ms.

A conclusão é de que o tempo médio é ligeiramente superior à primeira situação (376ms), apesar disso, a variação do tempo entre diferentes pedidos é menor, verificando-se maior estabilidade. Recorde-se que na primeira situação, para intervalos de leitura de 500ms, houve pedidos com tempos muito baixos, na casa dos 20ms, que correspondiam às situações em que a aplicação estava ocupada, o que contribuiu para um tempo médio de 376ms.

A última situação, com a aplicação local a vigiar a base de dados com intervalos de 200ms, mostra-se equilibrada e com um tempo médio de execução dos pedidos razoável.

Neste ponto pode-se considerar estar a trabalhar próximo de dados em tempo real, tendo em conta que os tempos de resposta são baixos. Na base destes resultados estão o facto dos pedidos serem executados por aplicações cliente, remotas, e de serem necessários vários passos para a conclusão de um pedido - consultas e várias alterações à base de dados, aplicação cliente e local e a comunicação entre a aplicação local e o equipamento.

Pode-se considerar que todo o sistema e arquitectura implementada são funcionais o suficiente e permitem uma quase completa interacção entre aplicações cliente remotas e os equipamentos.

Deve ter-se em atenção que nos casos analisados anteriormente a aplicação cliente e local estavam em computadores na mesma rede local, contribuindo positivamente para os tempos de transferência dos dados, portanto será de esperar que para interacções entre aplicações em locais diferentes os tempos aumentem ligeiramente.

Outro ponto a ter em conta é o tempo de processamento dos dados pelo computador, que influencia directamente o tempo de “consumo” de um *Web service*, ou seja de acordo com o servidor existente, e a velocidade de processamento do mesmo, conseguir-se-ão tempos diferentes.

Para situações com mais do que um cliente a aceder ao mesmo *Web service*, os ensaios que foram feitos mostram que o tempo de consumo de um serviço se mantém, embora o tempo real para um serviço possa ser maior, algo que acontece sempre que a aplicação local estiver ocupada, ou seja, se aplicação estiver a processar um pedido do cliente 1, o cliente 2 terá, obrigatoriamente, de esperar pela conclusão do serviço em curso, aumentando assim o tempo real para o consumo do serviço, desde o momento em que este é pretendido.

6 Conclusões e trabalhos futuros

Na presente dissertação foi abordado um sistema que permite a supervisão e controlo de vários equipamentos através da *Web*. Uma qualquer aplicação cliente, remota, podendo inclusive ser uma aplicação a correr num *PDA* ou simplesmente através de um *Internet Browser*, possibilitar comunicar de forma indirecta com um ou mais equipamentos.

No primeiro capítulo apresentou-se o contexto do problema, a crescente necessidade de acompanhamento da evolução, automatização e flexibilidade e a consequente necessidade de supervisão e controlo de processos e sistemas de produção local e remota, apontando-se os *Web services* como uma tecnologia que permite ultrapassar as barreiras da distância entre aplicações e equipamentos.

O problema levantado culminou numa ampla pesquisa relativa a questões como arquitectura e tecnologia dos *Web services*, modos de implementação, características, protocolos dos *Web services* e estado de arte. Em estudo e análises estiveram também sistemas que permitam supervisão e controlo, ou seja, sistemas e softwares *SCADA*, modelos de transmissão de dados, sistemas baseados na tecnologia *OPC*, especificações *OPC, OPC Servers* e respectivos estados de arte.

Na sequência do estudo de várias soluções propostas, é demonstrado todo o desenvolvimento e implementação do sistema criado, desde a arquitectura à implementação dos serviços, apresentando-se um sistema do tipo cliente-servidor com uma arquitectura descentralizada e orientada a serviços, que possibilita a integração de serviços em aplicações remotas e a monitorização e controlo remoto de equipamentos.

Qualitativamente e numa análise comparativa com os servidores *OPC*, o sistema criado revela características e valências idênticas às de um servidor *OPC*, apresentando, contudo, a vantagem de permitir a supervisão e controlo remoto de equipamentos, de forma completa e pelo facto de se tratar de uma solução, na sua grande parte, *freeware.*

Analisados os tempos de execução e resposta dos serviços criados, constatou-se que os tempos médios foram significativamente aceitáveis, tendo em conta todos os passos necessários para a invocação de um serviço e obtenção da resposta. O tempo médio da execução completa de um serviço é inferior a 400ms, o que coloca os dados próximos de dados em tempo real.

O sistema desenvolvido pauta-se por um conjunto de *Web services* desenvolvidos de acordo com a especificação *OPC XML DA* e de acordo com as características mais comuns de alguns servidores *OPC* e softwares *SCADA.*

O sistema é composto, necessita apenas de uma aplicação local que processa os pedidos e permite a comunicação com vários equipamentos. A interacção entre aplicação local e cliente foi desenvolvida segundo a especificação *OPC XML DA*, realizada através da base de dados do *MySQL*, a qual tem uma estrutura definida e igual entre equipamentos.

Para permitir a comunicação com os equipamentos, a aplicação local contém as rotinas necessárias para o devido efeito. Apenas para os *PLC's* da *Mitsubishi* foi utilizada uma biblioteca dinâmica da *Mitsubishi,* todavia esta não implica a instalação de *software,* as restantes foram desenvolvidas ou integradas com objectos *freeware*. Todas as rotinas de comunicação mostraram-se capazes e fiáveis o suficiente, permitindo aceder a todas as áreas de memória dos equipamentos.

Pode-se considerar que o sistema desenvolvido corresponde às expectativas, tendo conseguido atingir os principais objectivos definidos, estando pronto para trabalhos futuros, no sentido de aumentar a sua eficiência e permitir a interacção com mais equipamentos, para além de estar apto a explorar outros meios de comunicação.

Trabalhos Futuros…

No término da dissertação foi possível constatar que, apesar do sistema implementado se mostrar útil, intuitivo, de fácil integração e fiável, correspondendo às expectativas iniciais, ainda há muito para desenvolver, como:

- A integração de protocolos de comunicação com mais equipamentos;
- Preparar o sistema para interagir com sistemas OPC e activeX;
- Explorar outros meios de comunicação de dados, para além da porta série, tais como USB, etc;
- Explorar outras tecnologias de desenvolvimento, tais com JAVA, que permitem outras potencialidades e formas de interacção;
- Permitir o acesso de vários clientes, em simultâneo, ao mesmo equipamento.
- Integração em ambientes industriais;

Para além dos pontos referidos, será importante analisar a possibilidade de integrar um sistema do género (um sistema de monitorização e controlo remoto que permita a interoperabilidade entre sistema, abstracção dos clientes das camadas mais inferiores) a outras realidades e recursos. Em causa pode estar o estudo da integração com sistemas AVAC (Aquecimento Ventilação e Ar Condicionado), de forma a garantir maior eficiência destes sistemas e permitir a interacção.

A integração com sistemas de micro-geração será mais uma potencial área de análise, para que, de acordo com as condições existentes ou de acordo com as preferências dos clientes, se possa definir modos de funcionamento e permitir maior eficiência energética.

Surge ainda a possibilidade de vir a dotar um sistema, semelhante ao implementado descrito na dissertação, de tomada de decisão, ou seja, permitir que em função de eventuais defeitos ou deficiências, o sistema tenha capacidade de resolução e melhorias para o bom funcionamento de um recurso ou equipamento.

Para as situações descritas, como será natural, será necessário o desenvolvimento de novos serviços…

7 Bibliografia

[Advosol, 2008]
Advosol. 2008. Advosol Inc. [Online] 2008. http://www.advosol.com/default.aspx.

[Alecrim, 2006]
Alecrim, Emerson. 2006. Banco de dados MySQL e PostgreSQL. *Info Wester.* [Online] Novembro de 2006. http://www.infowester.com/postgremysql.php.

[Macedo, 2004]
Aplicações Distribuídas na Internet. Macedo, Rodrigo C., et al. 2004. Lisboa : s.n., 2004. 5º Conferência da Associação Portuguesa de Sistemas de Informação.

[Campos, 2006]
Campos, Filipe. 2006. *Estudo de arquitecturas e desenvolvimento de ferramentas em plataformas Mobile para sistemas Scada.* [Tese] Porto : Faculdade de Engenharia da Universidade do Porto, 2006.

[Chitnis, 2002]
Chitnis, Mandar, Tiwari, Pravin e Ananthamurthy, Lakshmi. 2002. Introduction to Web Services Part 2: Architecture. *developer.com.* [Online] 5 de Novembro de 2002.

[Clarke, 2004]
Clarke, Gordon, Reynders, Deon e Wright, Edwin. 2004. *Practical Modern SCADA Protocols.* Burlington : Elsevier, 2004.

[Fonseca, 2002]
Comunicação OPC – Uma abordagem prática. Fonseca, Marcos. 2002. Vitória, Brasil : VI Seminário de Automação de Processos, 2002.

[COPA-DATA, 2009]
COPA-DATA. 2009. *Copa-Data - Zenon SCADA/HMI.* [Online] 2009. http://www.copadata.com/.

[Costa, 2007]
Costa, Diogo; Rito, Rui. 2007. SOFTWARE DE SUPERVISÃO E CONTROLO DE. Aveiro : Universidade de Aveiro, 2007. Projecto.

[Costa, 2007b]
Costa, Diogo; Rito, Rui. 2007. SOFTWARE DE SUPERVISÃO E CONTROLO DE. Aveiro : Universidade de Aveiro, 2007. Projecto.

[Costa, 2006]
Costa, Diogo; Rito, Rui. 2006. Supervisão e controlo de dispositivos via Web com recurso a servidores OPC. Aveiro : Universidade de Aveiro, 2006. Relatório.

[Cunha, 2002]
Cunha, Davi. 2002. Web Services, SOAP e Aplicações Web. *Netscape devedge.* [Online] 10 de Dezembro de 2002.

[DeltaV, 2007]
DeltaV. 2007. *OLE for Process Control (OPC) Overview.* [Documento] s.l. : Emerson Process Management, 2007.

[DEEI, 2007]
Departamento de Engenharia Electrónica e Informática. 2007. Aplicações Distribuídas na Internet. [Online] 2007. http://intranet.deei.fct.ualg.pt/ADI/webservices.pdf?q=ADI/.

[DEETC, 2005]
Departamento de Engenharia, Electrónica, Telecomunicações e Computadores. 2005. Modelo TCP/IP. Lisboa : Instituto Superior de Engenharia de Lisboa, 2005. Apresentação.

[DEETC, 2008]
Departamento de Engenharia, Electrónica, Telecomunicações e Computadores.. 2008. Redes de Comunicação - Modelo OSI. Lisboa : Instituto Superior de Engenharia de Lisboa, 2008. Apresentação.

[Elipse, 2008]
Elipse Software. 2008. *Elipse Software.* [Online] 2008. http://www.elipse.com.br/.

[Exata, 2008]
Exata, sistemas de automação. 2008. MoviconX Técnico. *Exata, sistemas de automação.* [Online] 2008. http://www.moviconx.com.br/page.aspx.

[Fagor, 1999]
Fagor Automation – CNC 8050/55M Manual de Programação. 1999.

[FEUP, 2002]
FEUP. 2002. IP - Arquitectura, Interligação, Encaminhamento, Evoluções. Porto : Faculdade de Engenharia do Porto, 2002. Curso.

[Gomez, 2002]
Gomez, Jose Angel Gomez. 2002. *Survey of SCADA systems and visualization of a real life process.* [Tese] LINKÖPING : Department of Electrical aEngineering, Linköping University, 2002.

[Inoxnet, 2004]
Inoxnet. 2004. *API e Integração.* [Documento] Lisboa : Inoxnet, 2004.

[Kepware, 2008]
Kepware Technologies. 2008. OPC - OLE for Process Control Overview. *Kepware Technologies.* [Online] 2008. http://www.kepware.com/Menu_items/industry_OPC_Foundation.html.

[Kepware, 2008b]
Kepware Technologies. *kepware Technologies.* [Online] 2008. http://www.kepware.com/.

[Krutz, 2006]
Krutz, Ronald L. 2006. *Securing SCADA Systems.* Indianapolis : Wiley Publishing, Inc., 2006.

[Lange, 2006]
Lange, Jürgen. 2006. Quo Vadis OPC? From Data Access to Unified Architecture. [Online] 2006. http://www.softing.com/home/.

[Lopes, 2005]
Lopes, Carlos Jorge e Ramalho, José Carlos. 2005. *Web Services: Aplicações distribuidas sobre protocolos internet.* Lisboa : FCA - Editora de informática, 2005.

[Maciel, 2003]
Maciel, Paulo Henrique Soares. 2003. *RT 010.03 - Elipse.* [Documento] Brasil : Elipse Software, 2003.

[Mackay, 2004]
Mackay, Steve, et al. 2004. *Industrial Data Networks.* Burlington : Elsevier, 2004.

[Mahnke, 2009]
Mahnke, Wolfgang, Leitner, Stefan-Helmut e Damm, Matthias. 2009. *OPC Unified Architecture.* s.l. : Springer-Verlag Berlin Heidelberg, 2009. 978-3-540-68898-3.

[MELSOFT, 2002]
MELSOFT Integrated FA Software. 2002. *MX Component V3 - Programming Manual.* [Manual de Programação] s.l. : MITSUBISHI ELECTRIC CORPORATION, 2002. SW3D5C-ACT-E.

[Mitsubishi, 2003]
Mitsubishi Electric. 2003. *MX OPC Server Quick Start.* [Manual] s.l. : Mitsubishi Electric, 2003. UG-MXO-101.

[National Instruments, 2009]
National Instruments. 2009. NI LookOut. *National Instruments.* [Online] 2009. http://sine.ni.com/nips/cds/view/p/lang/en/nid/12511.

[OMRON, 2006]
OMRON. 2006. *Communications Commands Reference Manual.* [Documento] s.l. : OMRON, 2006.

[OMRON, 2008]
OMRON. 2008. Industrial Automation. *OMRON.* [Online] 2008. http://industrial.omron.eu/en/products/catalogue/automation_systems/software/runtime/cx-supervisor/default.html.

[OMRON, 2009]
OMRON. 2009. *CX OPC Server The Open Industry Standard applied.* [Documento] s.l. : Omron, 2009.

[OPC, 2003]
OPC Foundation. 2003. *OPC Data Access Custom Interface Specification 3.0.* [Documento] s.l. : OPC Foundation, 2003.

[OPC, 2004]
OPC Foundation. 2004. *OPC XML-DA Specification 1.01.* [Documento] s.l. : OPC Foundation, 2004.

[OPC, 2008]
OPC Foundation. 2008. *OPC Unified Architecture Specification Overview and Concepts.* [Documento] s.l. : OPC Foundation, 2008.

[OPC Task Force, 1998]
OPC Task Force. 1998. OPC Overview. [Documento]. s.l. : OPC Foundation, 1998. Vol. 1º, Industry Standard Specification.

[OPCTI, 2007]
OPC Training Institute. 2007. OPC Specification. *OPC Training Institute.* [Online] 2007. http://www.opcti.com/OPC-Specifications.aspx.

[Pamplona, 2009]
Pamplona, Vitor Fernando. 2009. *Web site de JavaFree.org.* [Online] 28 de Agosto de 2009. http://javafree.uol.com.br/artigo/871485/.

[Pereira, 2004]
Serviços Web. Pereira, Óscar Narciso Mortágua. 2004. Aveiro : s.n., 2004.

[Pereira, 2005]
Pereira, Carlos Ângelo e Lopes, João Correia. 2005. *Interacção com uma Arquitectura de Componentes com Independência de Dispositivo e de Localização.* [Tese] Porto : Faculdade de Engenharia da Universidade do Porto, 2005.

[Quintã, 2008]
Quintã, André Figueiredo. 2008. *Integração de Sistema de Produção.* [Tese] Aveiro : Departamento de Engenharia Mecânica, Universiade de Aveiro, 2008.

[Quintas, 2004]
Quintas, António Rocha. 2004. *SCADA.* [Documento] Porto : Faculdade de Engenharia da Universidade do Porto, 2004.

[Rogado, 2008]
Rogado, José e Gama, Pedro. 2008. *Computação Distribuída – Cap. VI.* [Documento] Lisboa : Universidade Lusófona, 2008.

[Shklar, 2003]
Shklar, Leon e Rosen, Richard. 2003. *Web Application Architecture - Principles, protocols and practices.* Chichester : John wiley & Sons, Ltd, 2003.

[Siemens AG, 2007]
Siemens AG. 2007. Simatic Wincc. *Industry automation and drive technologies - Siemens.* [Online] 2007. http://www.automation.siemens.com.

[Silva, 2007]
Silva, Rafael Marques da e Martino, Renan Correia. 2007. *Integração de Aplicações de Grid ao Globus 4.* Brasília : Universidade de Brasília, 2007.

[Ingear, 2006]
Software Toolbox. 2006. INGEAR OPC Server. [Online] Software Toolbox, 2006. http://www.ingearopc.com/.

[Sourceforge]
Sourceforge. LIBNODAVE -- Exchange data with Siemens PLCs. *sourceforge.net.* [Online] [Disponível: 2008.] http://libnodave.sourceforge.net/ .

[Souza, 2006]
Souza, Alessandro J. de e Filho, Francisco Sales de L. 2006. *SUPERVISÃO DE PROCESSOS INDUSTRIAIS USANDO WEB SERVICE.* Natal - Brasil : s.n., 2006.

[Sumra, 2003]
Sumra, Rajesh. 2003. Developing JAX-RPC–Based Web Services Using Axis and SOAP. [Online] 19 de Julho de 2003. http://www.developer.com/java/web/article.php/2237251/Developing-JAX-RPCndashBased-Web-Services-Using-Axis-and-SOAP.htm.

[Tech-faq, 2008]
Tech-faq. 2008. What is SCADA? *tech-faq.* [Online] Janeiro de 2008. http://www.tech-faq.com/scada.shtml.

[Urzêda, 2006]
Urzêda, Claiton Cesar de. 2006. *Software SCADA como plataforma para a racionalização inteligente de energia eléctrica em automação predial.* [Tese] Brasília : FACULDADE DE TECNOLOGIA, Universidade de Brasília, 2006.

[Lopes, 2004]
Web Services: Metodologias de Desenvolvimento. Lopes, Carlos J. Feijó e Ramalho, José Carlos. 2004. Porto : Reprografia da UM, 2004.

[Wikipedia, 2009]
Wikipedia. 2009. Modelo OSI. *Wikipedia.* [Online] 2009. http://en.wikipedia.org/wiki/OSI_model.

[Wikipedia, 2009b]
Wikipedia. 2009. MySQL. *Wikipedia.* [Online] 2009. http://pt.wikipedia.org/wiki/MySQL.

[Wikipedia, 2008]
Wikipedia. 2008. Scada. *Wikipedia.* [Online] 2008. http://en.wikipedia.org/wiki/SCADA.

[Wikipedia, 2008b]
Wikipedia. 2008. OLE for process control. *Wikipedia.* [Online] 2008. http://en.wikipedia.org/wiki/OLE_for_process_control.

[Wikipedia, 2008c]
Wikipedia. 2008. Universal Description Discovery and Integration. *Wikipedia.* [Online] Wikipedia, 2008. http://en.wikipedia.org/wiki/Universal_Description_Discovery_and_Integration.

[Zampronha, 2008]
Zampronha, Rogério. 2008. A Evolução dos Sistemas Supervisórios. *SoftBrasil.* [Online] Grupo SoftBrasil, 2008. http://www.softbrasil.com.br/site/artigos/acao:ver/indice:117.

Anexos

A1 Protocolos dos Web Services

Os Web services baseiam-se, como é descrito no capítulo anterior, principalmente em quatro protocolos standard a correr sobre o Protocolo HTTP, portanto, independentemente da linguagem em que seja elaborado o Web Service, este pode ser sempre interpretado por qualquer outra linguagem. Segue a descrição dos protocolos XML, SOAP, WSDL e UDDI que compõem os Web Services e o protocolo HTTP que é a base da comunicação WEB.

XML

É um subtipo de *SGML* (acrónimo de *Standard Generalized Markup Language*, ou Linguagem normalizada de Marcação Genérica) capaz de descrever diversos tipos de dados, com um propósito principal que se pauta pela facilidade de compartilhar informações através da Internet. Entre as linguagens baseadas em *XML* incluem-se *XHTML* (formato para páginas Web), *MathML* (formato para expressões matemáticas), *SVG* (formato gráfico vectorial), entre outros [Shklar, 2003].

Características do *XML:*

Estimulado pela insatisfação com os formatos existentes (normalizados ou não), o *World Wide Web Consortium (W3C)* começou a trabalhar em meados da década de 90 numa linguagem de marcação que combinasse a flexibilidade da SGML com a simplicidade da *HTML*. O princípio do projecto era criar uma linguagem que pudesse ser lida por software e integrar-se com as demais linguagens. Uma linguagem com uma filosofia que visava incorporar vários princípios:

- Separação do conteúdo da formatação;
- Simplicidade e Legibilidade, tanto para humanos quanto para computadores;
- Possibilidade de criação de tags sem limitação;
- Criação de arquivos para validação de estrutura (Chamados *DTDs*);
- Interligação de bancos de dados distintos;
- Concentração na estrutura da informação, e não na sua aparência.

O *XML* é considerado um bom formato para a criação de documentos com dados organizados de forma hierárquica, como se vê frequentemente em documentos de texto formatados, imagens vectoriais ou bancos de dados [Shklar, 2003].

Pela sua portabilidade, um banco de dados pode, através de uma aplicação, escrever num arquivo *XML*, e um outro banco distinto pode ler estes mesmos dados.

SOAP

O *Simple Object Access Protocol (SOA*P) é um protocolo para troca de mensagens, sendo que a proposta inicial do SOAP era a invocação de procedimentos remotos transportados via *HTTP*, porém o *SOAP* é flexível e pode ser utilizado com outros protocolos de transporte. O SOAP é projectado para invocar aplicações remotas através de *RPC (Remote Procedure Calls* - Chamadas Remotas de Procedimento) ou trocas de mensagens, num ambiente independente de plataforma e linguagem de programação. *SOAP* é, portanto, uma norma normalmente aceite para se utilizar com *Web Services*. Desta forma, pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização de uma linguagem (*XML*) e mecanismo de transporte (*HTTP*) normalizado.

O *SOAP* pode ser entendido como um dos pilares da arquitectura *Web Service*, proporcionando uma fácil comunicação entre diversas aplicações. Actualmente, existe uma grande estrutura de suporte ao protocolo e implementações em várias linguagens, além de vários serviços *SOAP* disponíveis na *Web* [Shklar, 2003].

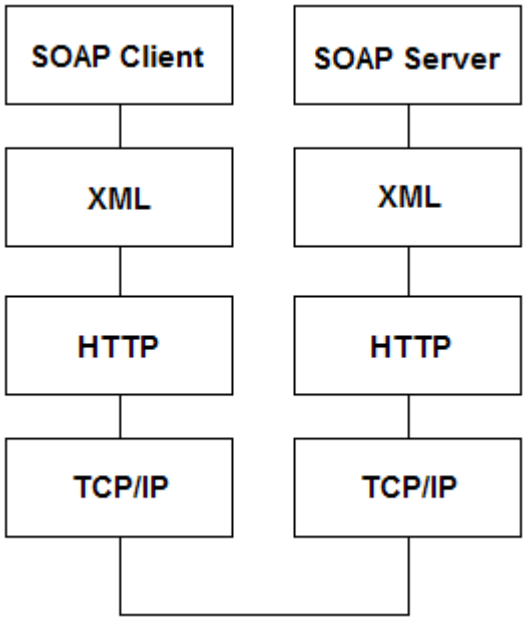


Figura 49 - Estrutura da comunicação de mensagens sob o protocolo *SOAP* [Rogado, 2008].

Mecanismos oferecidos pelo protocolo *SOAP*:

- Mecanismo para definir a unidade de comunicação;
- Mecanismo para lidar com erros;
- Mecanismo de extensão que permite evolução;
- Mecanismo entre as mensagens *SOAP* e o *HTTP*, representar tipos de dados em *XML*.

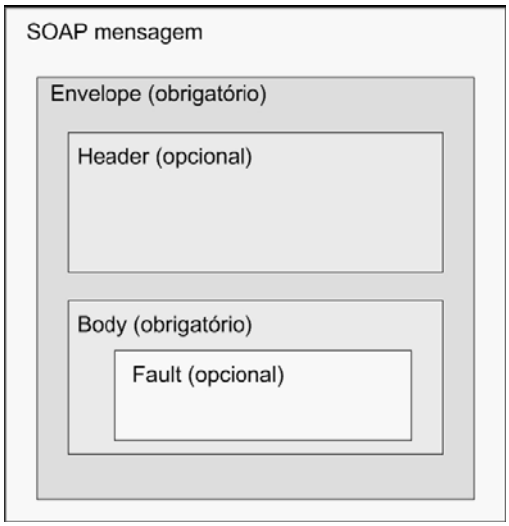


Figura 50 - Estrutura do protocolo *SOAP* [Silva, 2007].

A Figura 50 ilustra os principais elementos da mensagem *XML SOAP*, onde os elementos Envelope e *Body* são obrigatórios e os elementos *Header* e *Fault* são opcionais.

A especificação do protocolo *SOAP* pode ser definida em três partes, as mais relevantes e significativas [Silva, 2007], [Shklar, 2003]:

- *SOAP* Envelope: contém as especificações da aplicação com o nome do método a ser chamado remotamente;
- Regras de codificação de Dados: regras específicas para a troca de dados, de acordo com o seu tipo. O *SOAP* tem as suas próprias especificações baseadas na *W3C XML Schema specification*.
- *RPC* conventions: o *SOAP* possui normas para os vários sistemas de mensagens em que o protocolo é usado. As normas definem quais as informações a serem transmitidas e como devem ser definidas para a chamada remota do procedimento e respostas. Dados como o número de parâmetros que devem constar no documento, permitem a aplicação cliente receber a resposta do servidor.

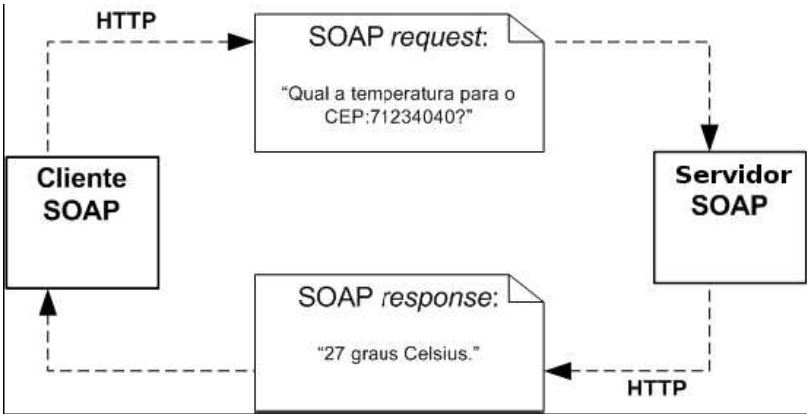


Figura 51 - Exemplo de troca de mensagens sobre o protocolo *SOAP* [Silva, 2007].

WSDL

A *WSDL (Web Service Definition Language*) é uma linguagem baseada na gramática *XML* e que define os serviços de um *Web Service* com a interface de uso, o tipo de dados que serão trocados no envio e recepção das mensagens, informação sobre qual protocolo a que o serviço está associado (geralmente é o protocolo *SOAP*) e o endereço onde se pode encontrá-lo, estabelecendo uma espécie de contratos entre cliente e servidor.

Existem seis tipos de dados básicos que estão na estrutura de um *WSDL* (Figura 52):

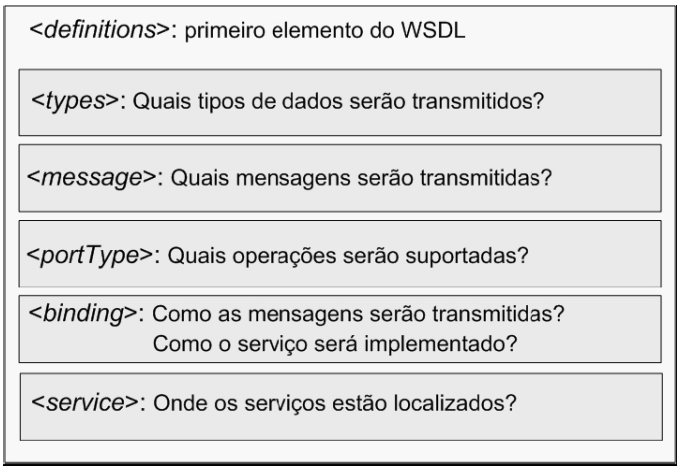


Figura 52 - Estrutura do *WSDL* [Silva, 2007].

Definitions é o primeiro elemento que inicia qualquer documento, e que define o nome do *Web Service* e todos os namespaces utilizados para definir certos padrões ao restante do documento.

Message define o nome da mensagem de enviada pelo cliente e o nome da resposta dada pelo servidor, utilizando o elemento *part* para determinar os nomes das variáveis que servirão de parâmetros para as operações ou para guardar os valores retornados. O elemento *type* diz qual é o tipo das variáveis.

PortType é uma combinação de vários elementos *message* informando qual a mensagem de ida, pelo elemento *input* e qual é a de volta, definido pelo elemento *output*.

Binding informa qual é o protocolo de troca de mensagens que será utilizado para transmitir as solicitações e respostas.

Services informa qual o endereço onde se encontra o *Web Service*.

[Shklar, 2003]

UDDI

UDDI (Universal Description, Discovery and Integration) é um protocolo aprovado como padrão pela *OASIS*. O *UDDI* especifica um método para publicar e descobrir directórios de serviços numa arquitectura orientada a serviço (*SOA*), sendo o Serviço de nomes e de directório para acesso a informação genérica sobre Web Services [Wikipedia, 2008c].

O *UDDI* pode ser considerado como uma base de dados de informações sobre *Web services*. Os fornecedores de serviços podem utilizar *UDDI* para publicar os serviços que oferecem, enquanto os utilizadores de serviços podem usar o protocolo para descobrir serviços que lhes interessem e obter os

métodos necessários para utilizar esses serviços. Actualmente existem diversas empresas a fazerem o *hosting* de *UDDI's* públicos.

A especificação UDDI define [Lopes 2005]:

- Serviços *SOAP* utilizados para publicar e obter informações de um registo *UDDI*;
- Esquemas *XML* do modelo de dados do registo e do formato das mensagens *SOAP*;
- Definições *WSDL* de serviços;
- Definições de registo UDDI (modelos técnicos - *tModels*) sistemas de identificação e especificação dos registos *UDDI*.

O Protocolo *UDDI* permite três tipos de acessos/pesquisas:

- Páginas Brancas (*White Pages*) – Nomes dos serviços e detalhes da localização;
- Páginas Amarelas (*Yellow Pages*) – Atributos e tipo de serviços;
- Páginas Verdes (*Green Pages*) – Informações Técnicas sobre os serviços.

O *UDDI* tem uma organização hierárquica em quatro níveis [Rogado, 2008]:

- *businessEntity*: características da organização que fornece os serviços, como actividade, nome, endereço, etc;
- *businessServices*: serviços agrupados por tipo designados por um nome e um ramo de actividade;
- *bindingTemplate*: endereço dos *Web Services* e referências para as suas descrições (por *URL*);
- *tModel*: descrição detalhada do serviço, geralmente fornecida por um ficheiro *WSDL*.

(Cada elemento (excepto para *tModel*) é identificado por uma chave e pode ser consultado directamente).

Interface de Publicação

- Permite registar um serviço e actualizar informação associada (Registos).

Registos

- Informação de uma dada instância de *UDDI* que é armazenada em registos contidos em vários servidores que podem ser replicados.

HTTP

O protocolo *HTTP (HyperText Transfer Protocol)* e também o *HTML (HyperText Markup Language)* resultaram de um projecto do *CERN (European Particle Physics Laboratory)*, em 1989. Estes padrões forneceram as bases para o aparecimento do *WWW (World Wide Web)* – o serviço de maior popularidade da Internet e grande responsável pelo seu actual crescimento [Shklar, 2003].

O *HTML* é um protocolo de comunicação (na camada de aplicação segundo o Modelo *OSI - Open Systems Interconnection*) utilizado para transferir dados em intranets e pela WWW.

Normalmente, este protocolo utiliza a porta 80 e é usado para a comunicação de sítios web, comunicando na linguagem *HTML*. Contudo, para haver comunicação com um servidor é necessário utilizar comandos adequados, que não estão na linguagem *HTML*.

O acesso a documentos faz-se através de um endereço de página da Internet – e para entrar no documento deve-se inserir o respectivo endereço, denominado *URI* (Universal Resource Indentifier), ou mais especificamente um *URL (Universal Resource Locator*), um tipo de *URI* que pode ser localizado directamente.

O protocolo *HTTP* faz a comunicação entre o cliente e o servidor através de mensagens. O cliente envia uma mensagem de requisição de um recurso e o servidor envia uma mensagem de resposta ao cliente com a solicitação. Os dois tipos de mensagem utilizam um formato genérico.

Uma mensagem, tanto de pedido quanto de resposta, é composta por uma linha inicial, nenhuma ou mais linhas de cabeçalhos, uma linha em branco obrigatória finalizando o cabeçalho e por fim o corpo da mensagem, opcional em determinados casos [Shklar, 2003].

Para que o protocolo *HTTP* consiga transferir os dados pela *Web*, é necessário que os protocolos *TCP* e *IP* (*Internet Protocol*, Protocolo de Internet) tornem possível a ligação entre clientes e servidores através de *sockets TCP/IP.*

A2 – Especificações OPC

OPC Data Exchange [OPC TI, 2007], [OPC Foundation, 2003b]

A especificação *OPC Data eXchange (OPC DX*) define a possibilidade da troca de dados entre servidores *OPC*. Actualmente, existem algumas companhias que incorporam esta especificação nos seus produtos, contudo, não há nenhuma aplicação comercial desta tecnologia, o que leva a que empresas utilizem várias formas de "pontes" *OPC* para transferirem dados entre os seus servidores *OPC*.

A *OPC Foundation* é responsável por manter e definir a especificação *OPC DX*. Para as restantes especificações foram definidas interfaces para os respectivos servidores que adquirem dados dos dispositivos de campo e transmitem-nos verticalmente até aos sistemas empresariais de supervisão.

Pelo contrário, a especificação *OPC DX* foi desenhada para mover os dados dos dispositivos de campo horizontalmente, entre servidores *OPC DA*, ou seja, uma especificação que transporta o conceito da comunicação Cliente/Servidor para a comunicação Servidor/Servidor. A nova tecnologia da *OPC* DX permite a interoperabilidade de dados entre sistemas baseados na tecnologia *OPC* (incluindo sistemas baseados na tecnologia *DCOM* e *XML* sobre *Ethernet*) abrangendo *PLC's*, *HMI*/*SCADA*, dispositivos E/S e computadores.

O propósito desta especificação passa pela continuidade do objectivo da tecnologia *OPC* para promover e permitir a interoperabilidade de aplicações e sistemas.

A *OPC DX* não especifica um novo método para transferência de dados, ao invés disso, deposita na *OPC DA* a capacidade de transferência de dados já em uso na actualidade.

OPC Complex Data [OPC TI, 2007], [OPC Foundation, 2003b]

A especificação *OPC Complex Data* (Dados Complexos) é um complemento da *OPC* DA e da *OPC XML DA* e permite aos servidores expor e descrever tipos de dados mais complexos, como estruturas binárias e documentos *XML*.

Por dados complexos (*Complex Data*) entende-se um termo que descreve itens da *OPC DA* com valores construídos. A especificação fornece os mecanismos para os clientes *OPC DA* descobrirem a estrutura dos valores dos dados.

Os itens da especificação *OPC DA* são variáveis de tipos simples, pelo que o principal objectivo da especificação *OPC Complex Data* é conseguir que os servidores apresentem dados estruturados de forma simples, como uma sequência de bytes. A especificação *DA* não dispõem dos mecanismos para os servidores descreverem a estrutura desses *bytes* e, como resultado, os clientes não têm conhecimentos sobre a estrutura e são incapazes de interpretar os dados.

A *OPC Complex Data* define a informação, representada como propriedades dos itens *OPC DA*, que os servidores *OPC DA* possam disponibilizar aos clientes para descrever a estrutura dos dados. Itens de dados construídos cuja estrutura é definida através destas propriedades são conhecidos como itens de dados complexos.

Um exemplo de dados complexos é a estrutura de dados que representa a ligação a dispositivos de Entradas e Saídas (E/S), a configuração de uma entrada ou saída de um dispositivo com formato apenas para leitura, informações de estado dos dispositivos e pontos de controlo que podem ser escritos.

OPC Commands [OPC TI, 2007], [OPC Foundation, 2003b]

A definição de parâmetros de interfaces que permitem aos clientes *OPC* e servidores identificar, enviar e monitorizar comandos de controlo que se executam num dispositivo, depende da especificação *OPC Commands.*

Dispositivos de automação (*PLC's*, controladores numéricos,...) por vezes definem certas funcionalidades através de comandos. Em comparação com variáveis (itens), os comandos têm um comportamento muito mais complexo:

- Cada comando tem uma definição de parâmetros, que podem ser variáveis;
- Cada comando pode ser executado várias vezes em paralelo;
- Um comando, por trás, tem um estado "máquina"; alterações ao estado podem ser notificadas;
- Os comandos podem retornar informação complexa após serem executados.

Os servidores *OPC Comands* podem ser implementados de forma independente ou em processo com outros servidores como *DA*, Alarmes e Eventos ou *Batch*.

Como finalidade da *OPC Commands* está o fornecimento de uma base comum para definições de comandos. Tendo por base esta especificação será possível o uso de clientes genéricos para executar e controlar comandos e é por esta razão que a *OCP Commands* tem um ênfase importante na forma de descobrir comandos e a execução associada aos estados máquina.

OPC Alarms and Events [OPC TI, 2007], [OPC Foundation, 2003b]

A especificação *OPC Alarmes and Events (A&E)* define interfaces entre clientes e servidores para definir, monitorizar e informar a existência de eventos e alarmes.

As interfaces fornecem os mecanismos para que os Clientes *OPC* possam ser notificados de ocorrências relativas a condições específicas de eventos e alarmes. Também disponibilizam serviços que

permitem aos Clientes OPS determinar quais as condições de eventos e alarmes suportadas por um Servidor *OPC* e saber qual o seu estado actual.

O conceito de eventos e alarmes é muito comum no processo de controlo industrial e por vezes com noções diferentes, daí a especificação *OPC A&E*. Contudo, em condições normais e para as especificações *OPC,* um alarme é uma condição anormal, ou seja, um caso especial de uma condição

Uma condição é um estado definido de um *OPC Event Server* ou de um dos seus objectos (*Tags*), que são do interesse do Cliente *OPC* e que podem ter as seguintes condições associadas: *HighAlarm, HighHighAlarm, Normal, LowAlarm, and LowLowAlarm.*

Um evento, por outro lado, é uma ocorrência detectável que é importante para Servidores *OPC*, dispositivos que estes representam e para clientes *OPC*. Um evento pode ou não estar relacionado com uma condição. Por exemplo, a transição entre condições de *HighAlarm* e *Normal* é um evento que está relacionado com uma condição. No entanto, acções de operadores, alterações nas configurações dos sistemas e erros de sistema são eventos que não estão relacionados com condições específicas.

Os clientes *OPC* podem escolher se recebem ou não notificações da ocorrência de determinados eventos.

Uma interface *IOPCEventServer* disponibiliza os meios que permitem ao cliente *OPC*:

- Determinar os tipos de eventos suportados pelo Servidor *OPC;*
- Fazer subscrições (*subscriptions*) de eventos específicos, para que os clientes *OPC* possam receber notificações das ocorrências dos mesmos. Podem também ser usados filtros para fazer uma divisão dos eventos desejados;
- Aceder e manipular condições criadas pelo Servidor *OPC*.

Os modelos *IOPCEventServer* podem suportar interfaces opcionais com o objectivo de endereçar condições implementadas pelo servidor e gerir grupos de condições públicas.

OPC Historical Data Access [OPC TI, 2007], [OPC Foundation, 2003b]

As bases de dados de históricos criaram uma fonte de informação adicional que deve ser distribuída por utilizadores e *software*s clientes que estejam interessados nesta informação. Actualmente, muitos dos sistemas de históricos usam as suas interfaces proprietárias para disseminação dos dados. Não há capacidades de aumentar ou usar soluções de históricos existentes com outras capacidades num ambiente *Plug-n-play*, uma vez que isso requer que o programador recrie as mesmas infra-estruturas para os seus produtos de forma a garantir alguma interoperabilidade.

A vontade e a necessidade de integrar os dados de todos os níveis leva a que a informação de históricos possa ser considerada como outro tipo de dados.

Existem vários tipos de servidores de histórico, alguns tipos chave suportados pela *OPC Historical Data Access*:

- Servidores de dados simples, que em geral, armazenam e dispõem dados básicos de um servidor *OPC Data Access*, normalmente na forma de Tempo, Valor e Qualidade.
- Dados comprimidos complexos e análises de servidores. Estes servidores fornecem compressão de dados assim como dados normais e são capazes de disponibilizar sumários de dados ou dados de análises de funções, tais como médias, valores máximos e mínimos, etc. Por outro lado, permitem também a actualização dos dados e a criação de históricos de actualizações. Para além disto permitem o armazenamento de anotações nas bases de históricos de dados.

O referido grupo de normas, criadas pela *OPC Foundation*, fornece especificações *COM* para comunicação de dados de dispositivos e aplicações que permitem o histórico de dados, tais como bases de dados. Desta forma, a especificação permite o acesso a todo o tipo de dados.

A especificação *OPC Historical Data Access*, ou simplesmente *OPC HDA*, é usada para a troca de dados armazenados, ao contrário da *OPC Data Access* que trata de dados em tempo real. A tecnologia *OPC* é também baseada na arquitectura Servidor/Cliente, ou seja, o cliente pode aceder ao servidor para que este lhe transmita o histórico de dados pretendido.

OPC Batch [OPC TI, 2007], [OPC Foundation, 2003b]

Uma outra especificação desenvolvida pela *OPC Foundation* é a *OPC Batch*, pensada para solucionar os problemas das produções por lotes.

A especificação *OPC Batch* define as interfaces para a troca de capacidades dos equipamentos e condições correntes de operação. A especificação foi definida para permitir que aplicações em *visual basic*, ou outras aplicações baseadas em automação, possam trocar dados de lotes com vários sistemas computorizados com pouca complexidade.

A *OPC Batch* tem como propósito fornecer os meios para transmitir dados e propriedades de lotes e equipamentos, em tempo real, entre equipamentos capazes de serem normalizados.

OPC Security [OPC TI, 2007], [OPC Foundation, 2003b]

A especificação *OPC Security* permite que as aplicações *OPC* consigam (*Apply*) uma segurança superior à disponível na tecnologia *DCOM*. A segurança *DCOM* permite que os servidores *OPC* definam o tipo de permissão de utilizadores ou grupos de correr ou aceder um servidor *OPC*.

A *OPC Security* possibilita que o administrador do servidor *OPC* permita ou recuse o acesso a determinados itens, o que confere às companhias o controlo do acesso dos clientes aos servidores, de forma a garantir que estes não acedem a informações para as quais não têm permissão e alterem os seus conteúdos ou mesmo definições do sistema.

A finalidade da especificação *OPC Security* é especificar como os servidores *OPC* devem implementar a segurança através dos recursos do sistema operativo, para além de que disponibiliza as instruções para a implementação *OPC* cliente interagir com a segurança do servidor *OPC*.

A3 - Objectos/ bibliotecas dinâmicas de comunicação

Objecto/biblioteca dinâmica Libnodave para comunicar com *Siemens* S7 200, S7 300 e S7 400 [Sourceforge]

O protocolo de comunicação com os autómatos da *Siemens* é o de mais difícil implementação, visto que correm em protocolos de transferência de dados desenvolvidos pela *Siemens*, são o MPI para as gamas S7-300 e S7-400 e o PPI para a gama S7-200. Outra dificuldade para implementar os protocolos de comunicação com os *PLC's* referidos, é a falta de informação disponível sobre estes protocolos, pois a *Siemens* apenas fornece informação para uso das suas próprias soluções.

Terminada a pesquisa foi possível encontrar um projecto Freeware para comunicação com autómatos da *Siemens*. A biblioteca Libnodave consiste em dois ficheiros DLL e possibilita a comunicação através de Portas Série, USB, e *TCP/IP*. Tem rotinas desenvolvidas para ler e alterar o estado de todas as variáveis e determinar os estados do *PLC*.

Para que seja possível ler ou alterar o estado das variáveis, é necessário efectuar alguns passos - abrir a porta de comunicação, efectuar a ligação com o cabo conversor MPI/RS232 (ou outro) e estabelecer a ligação ao *PLC*. Para terminar a ligação com o *PLC* implica que seja terminada a ligação pela ordem contrária - fechar a ligação ao *PLC*, terminar a ligação ao cabo e encerrar a porta de comunicação.

O uso das funções requer que seja adicionado o ficheiro *libnodave.net.dll* e colocado o ficheiro *libnodave.dll* na pasta Bin*Debug* do projecto em Visual Basic.Net. No passo seguinte definem-se as variáveis necessárias para o processo de ligação ao *PLC* , acesso às variáveis e estados do *PLC*.

Dimensionamento da variável para abrir porta de comunicação:

- *Dim fds As libnodave.daveOSserialType*

Dimensionamento da variável para estabelecer ligação com o cabo:

- *Dim di As libnodave.daveInterface*

Dimensionamento da variável para estabelecer ligação ao *PLC* e aceder às variáveis do *PLC*:

- *Dim dc As libnodave.daveConnection*

Dimensionamento da variável para obter o resultado da ligação ao cabo:

- *Dim res_cabo As Integer*

Dimensionamento da variável para obter o resultado da ligação ao *PLC*:

- *Dim res_PLC As Integer*

Dimensionamento da variável para obter o resultado da leitura ou escrita de uma variável:

- *Dim res As Integer*

Dimensionamento da variável para obter o estado de uma variável:

- *Dim a As Integer*

Dimensionamento da variável para o buffer interno:

- *Dim buf(1000) As Byte*

Funções necessárias e mais comuns:

Abrir porta de comunicação:

- *fds.rfd = libnodave.setPort("PORTA", "Baudrate", AscW("Paridade"))*
- *fds.wfd = fds.rfd*

O resultado desta operação é sempre maior que zero se tiver sucesso ao abrir a porta de comunicação.

Estabelecer ligação ao cabo:

- *di = New libnodave.daveInterface(variável para abrir porta, "Nome para a ligação", Endereço do cabo, Protocolo de comunicação, Baudrate do cabo e PLC)* – Definir ligação ao cabo.
  Exemplo: *di = New libnodave.daveInterface(fds, "I1", 0, libnodave.daveProtoMPI, libnodave.daveSpeed187k)*
- *di.setTimeout(1000000)* – Time-out para a tentativa de estabelecer ligação ao *cabo.*
- *res_cabo = di.initAdapter* – Iniciar ligação ao cabo

Estabelecer ligação ao *PLC*:

- *dc = New libnodave.daveConnection(Variável da ligação ao cabo, Endereço do PLC, Rack, Slot)* - Definir ligação ao *PLC*, Rack e Slot não interessam para os protocolos MPI e PPI.
  Exemplo: *dc = New libnodave.daveConnection(di, 2, 0, 0)*
- *res_PLC = dc.connectPLC()* – Iniciar ligação ao *PLC*

Verificar estado do *PLC*:

A seguinte função permite aceder a uma lista de identificação do *PL,* que, entre outras, está disponível em documentação da *Siemens,* sendo conhecidas por SZL.

Para receber os valores da lista é necessário definir um byte com dimensão igual ou superior ao número de valores que se pretendem ler.

- *Dim id(100) As Byte*
- *r = dc.readSZL(ID da lista, inicio, id, nº de valores a ler)*
  Exemplo: *r = dc.readSZL(25, 0, id, 100)*, na lista 25 é possível ler o estado leds de estado de funcionamento e diagnóstico do *PLC*, neste caso lêem-se 100 valores da lista 25, a começar na posição 0. Os valores são guardados um a um em cada posição do *byte id*.
  Na posição 14 do byte, *id(14)*, está o estado da luz de Run, na posição 18 o estado da luz de *Stop*, na posição 10 é o led do *System Failure*, posição 22 led do *FORCE* e na posição 26 o led do *Bus Failure*.

Leitura e alteraração do estado das variáveis:

Para ler ou alterar variáveis é importante lembrar que a *Siemens* define como bit mais significativo o primeiro, bit 0, e como menos significativo o bit 7.

Leitura:

- *r = dc.readBytes(Area, Nº DB , Byte Inicial, Nº de bytes a ler, buffer)*
  Onde, Área: indicação da variável; Nº DB: apenas para ler/escrever Data Blocks; Byte inicial: 1º Byte a ler; número de Bytes a ler; buffer: buffer interno para os dados.

Escrita:

- *r = dc.writeBytes(Area, Nº DB , Byte Inicial, Nº de bytes a ler, valores)*
  Neste caso os campos são iguais à leitura de bytes, à excepção do último, onde se colocam os valores a escrever.

Na tabela seguinte seguem-se as variáveis e a sua correspondência.

Tabela 10 - Áreas de memória e respectiva definição no objecto libnodave.

Nome da Área	Área constante	Exemplo de variável	Tipo
Data blocks	daveDB	DB3; DBD4	Bloco de dados
Marcas – variáveis de memória	daveFlags	MB4	Byte
Entradas	daveInputs	IB2	Byte
Saídas	daveOutputs	QB1	Byte

Temporizadores	daveTimer	T2	Word
Contadores	daveCounter	Z2	Word
Direct I/O	daveP	PIW4	Word
Informação de sistema no S7 200	daveSysInfo		Byte
Dados (variáveis de memória) no S7-200	daveDB	VB5	Byte
Marcas especiais no S7 200	daveSysFlags	SMB0	Byte
Entradas analógicas no S7 200	daveAnaIn	AIW0	Word
Saídas analógicas no S7 200	daveAnaOut	AQW0	Word
Temporizadores no S7 200	daveTimer200	T2	Word
Contadores no S7 200	daveCounter200	Z2	Word

Exemplo de Leitura de uma variável (byte MB4):

- *r = dc.readBytes(libnodave.daveFlags, 0, 4, 1, buf)*
- Para extrair o estado da variável é necessário fazer um get ao buffer interno:
  a = dc.getU8 – Lê um byte como "*Unsigned Value*", o valor vem sempre no formato decimal. Outras possibilidades para extrair o estado das variáveis são: como "*Signed Value*", *dc.getU8*; se pretender ler mais do que uma variável, por exemplo duas variáveis,
- *r = dc.readBytes(libnodave.daveFlags, 0, 4, 2, buf)* – pode extrair os valores byte a byte ou uma word. Para extrair byte a byte executa *a = dc.getU8,* para o primeiro e pode fazer logo de seguida outra, pois no buffer saltará directamente para o próximo byte, *a2 = dc.getU8*, ambos os valores em formato decimal.
- Para extrair uma word faz o get a 16 bits, ou seja, *a = dc.getU16*, sendo o valor igualmente em formato decimal.
- O *get* dos valores pode-se estender até 32 bits, *a = dc.getU32*, efectuando a leitura de quatro bytes.

Exemplo de alterar o estado de variáveis (byte MB4 e MB5):

Neste caso é necessário dimensionar uma variável *byte* com dimensão igual ao número de variáveis a alterar. Em cada posição do byte deve ir o valor no formato decimal para cada variável.

- *Dim valor(1) As Byte*

- *valor(0) = 3* – para activar os bits 0 e 1 do byte MB4, em binário 00000011.
- *valor(1) = 7* – para activar os bits 0, 1 e 2 do byte MB5.
- *r = dc.writeBytes(libnodave.daveFlags, 0, 4, 2, valores)*
  O resultado *r* é igual a 0 se a escrita dos valores for bem sucedida.

Ler e alterar um bit isolado

Ler estado:

- *r = dc.readBits(Área, Nº do Byte, Nº do bit, 1, buffer)*
  O campo área é igual ao da leitura de bytes, segue-se a identificação do Byte e depois a do bit (de 0 a 7). A seguir à identificação do bit deve surgir sempre o valor 1 e finalmente o buffer.
- Para extrair o estado: *a = dc.getU8* – Lê o valor do bit no formato decimal.

A função de leitura de bits isolados deve ser um recurso pouco utilizado, uma vez que pode gerar alguma confusão no CPU do *PLC*, pelo que deve ser usada a função para ler bytes e na resposta identificar o bit.

Alterar estado de um bit:

- *r = dc.writeBits(Área, Nº do Byte, Nº do bit, 1, Valor)*

Neste caso os campos são iguais à leitura de bits, à excepção do último, onde se coloca o estado a escrever.

Se a instrução for bem sucedida *r = 0*.

Contrariamente à função de leitura de bits, esta função deve ser utilizada quando se pretende alterar o estado de um único bit.

Fechar ligação ao *PLC*:

- *dc.disconnectPLC()*

Fechar ligação ao cabo:

- *di.disconnectAdapter()*

Fechar e limpar porta de comunicação:

- *libnodave.closePort(fds.rfd)*

Objecto/Biblioteca ActPcCom do *MX* Component da *Mitsubishi* [MELSOFT, 2002]

Para comunicar com os CPU dos *PLC* da *Mitsubishi*, mais concretamente as séries FX, A, QnA, QQ e QA recorreu-se a Biblioteca de Ligação Dinâmica (DLL) fornecida pela *Mitsubishi*. Esta DLL forma a comunicação pela porta série com o correspondente CPU do *PLC*. Para além da função de ligação (que tem como particularidade apenas a definição da porta COM) a DLL ActPcCom permite ler ou escrever dados da memória do *PLC*, alterar o estado do CPU, ler o tipo de CPU, ler os erros, etc.

A viabilidade de ler ou alterar o estado das variáveis requer que seja aberta a porta de comunicação e para terminar a comunicação com o *PLC* é imperativo que se feche a porta de comunicação.

O uso das funções torna necessário adicionar o ficheiro *ActPcCom.dll* e no passo seguinte criar o objecto que permite executar todas as funções.

Exemplo: criação do objecto para obter as funções para comunicar com a série FX:

Dimensionamento da variável para abrir porta de comunicação:

- *Dim FX As ACTPCCOMLib.ActFXCPU*

Para estabeler a comunicação com o *PLC* usa-se a função:

- *FX.ActPortNumber = porta de comunicação série*
- *FX.Open()*

O objecto criado tem uma função para *Timeout,* um tempo em que o VB vai tentar estabelecer contacto com o *PLC* e caso este contacto falhe gera-se uma excepção.

- *FX.ActTimeOut = time*

Para fechar a comunicação usa-se a função:

- *FX.Close()*

Ler ou escrever na memória de dados pode passar pelas funções Read e Write DeviceRandom respectivamente:

- *FX.ReadDeviceRandom (memórias, quantidade, valor)*
- *FX.WriteDeviceRandom (memórias, quantidade, valor)*

As memórias strings com os endereços de memória a ler ou escrever resultam numa quantidade que é o número de endereços que se pretende ler/escrever de acordo com as memórias definidas e o valor (Long), que fica registado depois de ler ou que é escrito na memória do *PLC*. A variável valor deve ser dimensionada de acordo com o número de dados a enviar ou receber.

- *Dim valor(quantidade) as Long*

Para além das funções *ReadDeviceRandom* e *WriteDeviceRandom,* existem também as funções *ReadDeviceBlock* e *WriteDeviceBlock*, entre outras não mencionadas, que têm o mesmo papel que as indicadas, sendo apenas diferentes na forma como acedem às memórias. As dissemelhanças entre as funções caracterizam-se apenas pela leitura/escrita em blocos sequenciais, desta forma é somente necessário indicar a memória inicial, a quantidade de memórias e a função lê/escreve sequencialmente, a partir da memória inicial definida.

Host-link [OMRON, 2006]

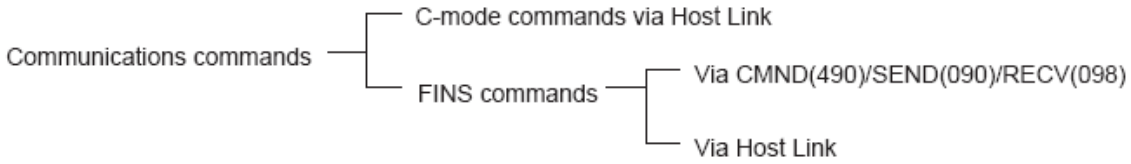
As comunicações Host-link são um tipo de protocolo de comunicações convencional, onde o *PLC* envia uma resposta consoante os comandos pedidos pelo computador remoto e podem ser usados para ler ou escrever dados nas áreas de dados do *PLC* e controlar algumas operações do *PLC*. Usando este tipo de comunicações não há necessidade de usar qualquer programa de comunicações no *PLC*.

Conforme os manuais sobre este tipo de comunicações, disponíveis no *Web* site da *Omron*, foi criada uma Biblioteca de Ligação Dinâmica (um ficheiro DLL). A par dos comandos que existem para desempenhar determinadas funções, foram desenvolvidas funções na DLL as quais permitem:

- Ler ou escrever dados das áreas de dados dos *PLC*;
- Ler ou alterar o estado do CPU;
- Ler e limpar os erros detectados na CPU;
- Ler o modelo do *PLC*.

Na DLL também foi criada uma rotina que permite executar um Frame Check Sequence nas mensagens que são enviadas e recebidas para/pelo *PLC*. A DLL permite comunicar via RS-232 com os modelos CPM, CJ e CS da *Omron*.

As Comunicações Host-link foram desenvolvidas pela *Omron* com a finalidade de ligar PC (Programmable Controllers) e computadores remotos via cabo RS232C, mas também que se pudesse controlar as comunicações do PC a partir do computador remoto. Normalmente o computador remoto envia um comando para o PC, que por sua vez envia automaticamente a resposta de volta. Em geral, existem dois tipos de implementação das comunicações Host-Link, em que uma é baseada nos comandos C-mode e a outra em comandos FINS.



Comandos C-mode

Os Comandos C-mode são especialistas em comandos de comunicações Host-Link, sendo emitidos pelo computador remoto e enviados para a unidade CPU.

As séries CPM**, CJ e CS permitem este tipo de comunicação.

Os Comandos C-mode formam um sistema de comando/resposta para comunicações série, proporcionando operações de controlo entre a unidade CPU e o computador remoto.

Estas operações incluem leitura e escrita da memória I/O, alteração dos modos de operação, etc.

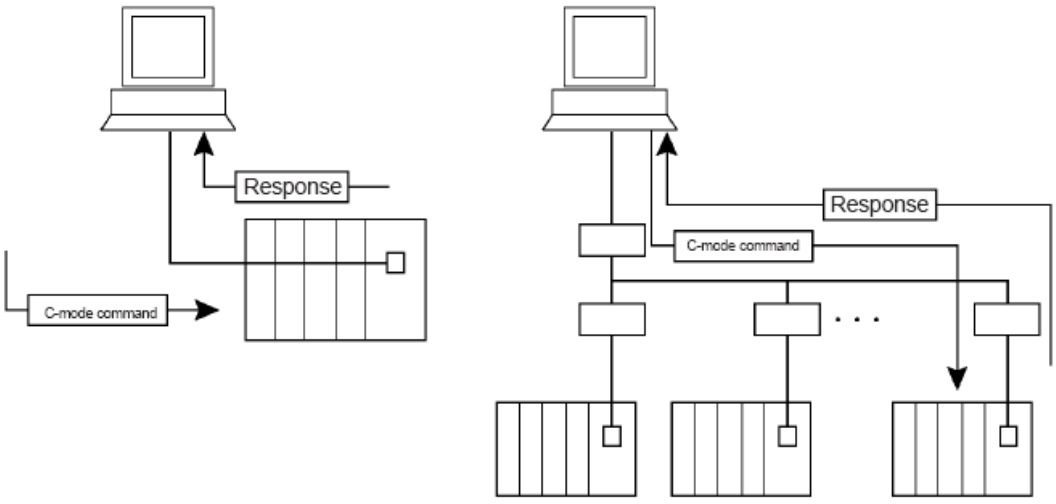


Figura 53 – Ligação de PLC's a computador remoto.

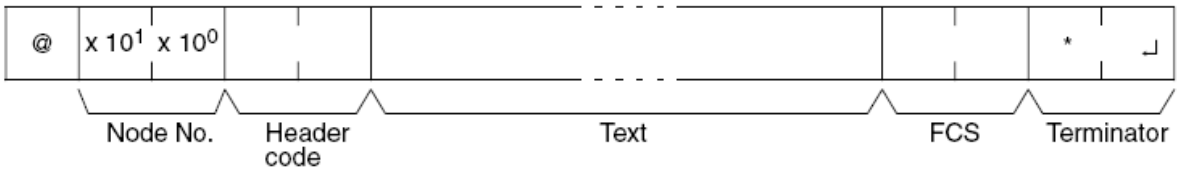
Até 32 *PLC*s (Unidades Host Link) podem ser ligados a um só computador remoto.

O comprimento de uma unidade de comando é chamada "frame", sendo que cada uma contem 131 caracteres de dados. Os caracteres são enviados em formato ASCII.

Formato de Comando/Resposta

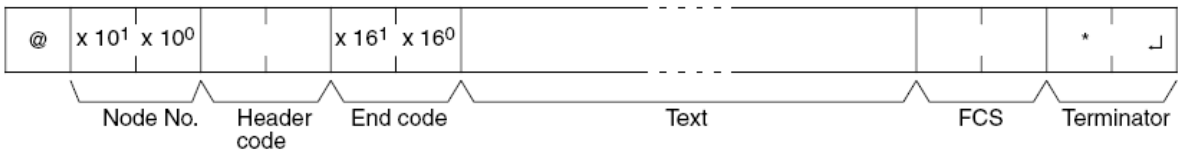
Se um comando não tiver mais de 131 caracteres de comprimento pode ser enviado numa só "frame".

Formato de Comando



- @: Informação de início de mensagem, tem de ser anexado no início do comando;
- Nó nº: Identifica o PC aos olhos do computador remoto. Este número tem de estar definido no PC setup;
- Código: Dois caracteres com o código do comando (ver Tabela 11);
- Texto: Parâmetros do código
- FCS: Dois caracteres para Frame Check Sequence (FCS);
- Terminação: Dois caracteres, "*" e o caracter(13) para indicar o fim do comando

Formato de Resposta



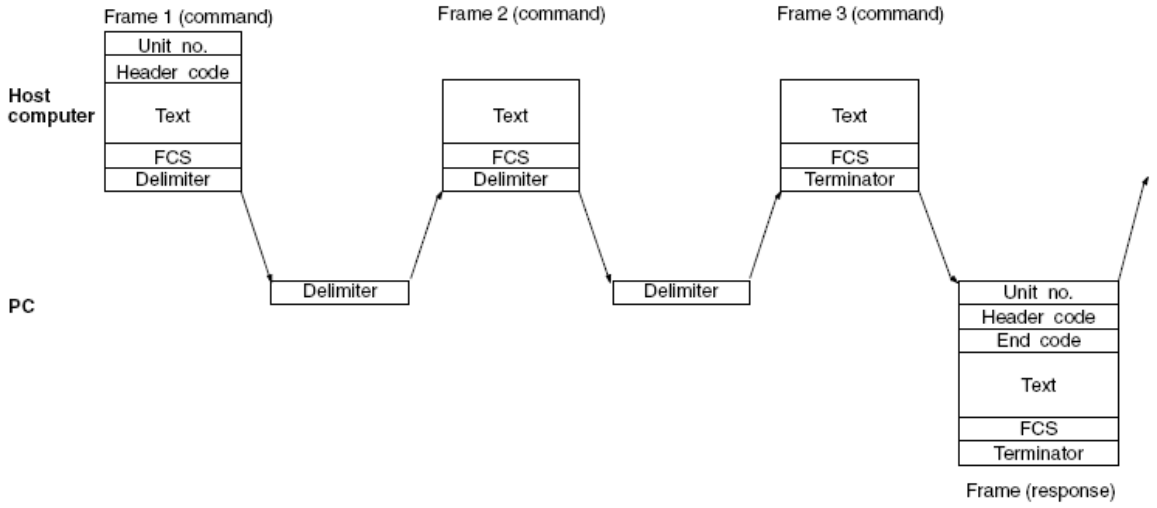
- @, Nó nº, Código: Devolvidos conteúdos idênticos ao comando.
- Código de Resposta: Nestes caracteres é devolvido o estado do comando recebido pelo PC (ou seja se existe ou não um erro na mensagem enviada).
- Texto: Devolvido texto quando o comando pedido foi do tipo leitura.
- FCS: Dois caracteres para Frame Check Sequence (FCS).
- Terminação: Dois caracteres, "*" e o caracter(13) para indicar o fim da resposta.

Transmissões Longas

Um longo bloco de dados pode ser transmitido tal como uma única "frame" de caracteres. Um comando ou uma resposta de 132 caracteres ou mais tem de ser dividido em mais do que uma "frame" antes de ser transmitido. Quando os dados são divididos, o fim da primeira e das "frames" intermédias é constituído apenas por um caracter (13) e não por um "*" e caracter (13).

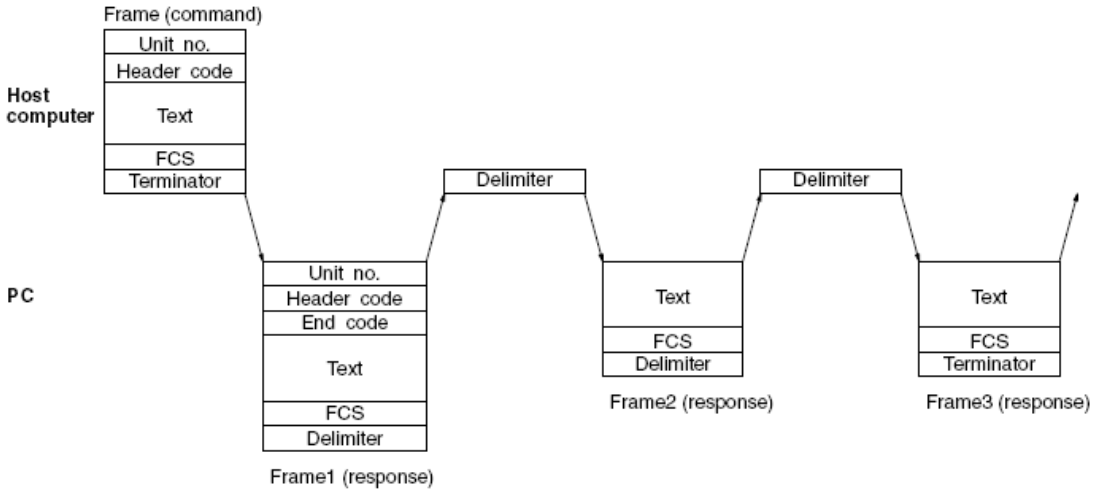
- Divisão de Comandos

Por cada "frame" transmitida pelo computador remoto, o computador espera por um caracter (13) que é transmitido pelo PC. Depois de recebido este caracter (13), a próxima "frame" vai ser transmitida. O procedimento repete-se até que a totalidade do comando seja transmitida.



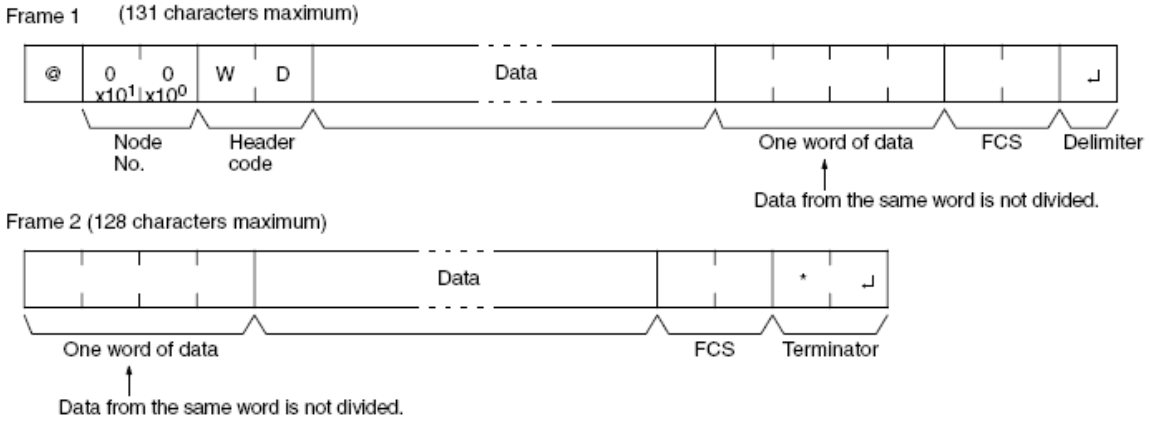
- Divisão de Repostas

Por cada "frame" recebida pelo computador remoto, é enviado um caracter (13) para o PC, seguindo-se a transmissão da próxima "frame". O procedimento é repetido até a totalidade da resposta ser transmitida.



- Precauções em transmissões longas

Quando se dividem comandos como WR, WL, WC ou WD que executam operações de escrita, deve-se prestar atenção para que não se dividam dados em "frames" separadas, que são escritas numa única word. Tal como ilustrado abaixo, a divisão de "frames" deve coincidir com a divisão entre words.



FCS (Frame Check Sequence)

Quando uma "frame" é transmitida, uma FCS é colocada antes da terminação de forma a que a mensagem seja analisada e seja determinado se existiu um erro na transmissão dos dados. O FCS é um byte convertido para dois caracteres ASCII. Este byte é o resultado de um XOR entre todos os caracteres entre o início da "frame" e o final do texto da "frame". Calculando o FCS sempre que uma mensagem é recebida e comparando este valor com o do FCS recebido na "frame", é possível verificar se a mensagem chegou com erros.

	ASCII code		Leftmost		Rightmost
@	→ 40	→	0100		0000
				XOR	
1	→ 31	→	0011		0001
				XOR	
0	→ 30	→	0011		0000
				XOR	
R	→ 52	→	0101		0010
				XOR	
0	→ 30	→	0011		0000
				XOR	
0	→ 30	→	0011		0000
				XOR	
to					
0	→ 30	→	0011		0000
				XOR	
1	→ 31	→	0011		0001
Calculation results			0100		0010
			↓	↓	Converted to hexadecimal.
			4	2	Handled as ASCII characters.

Tabela 11 - Códigos e sua definição, associados às memórias de leitura e escrita.

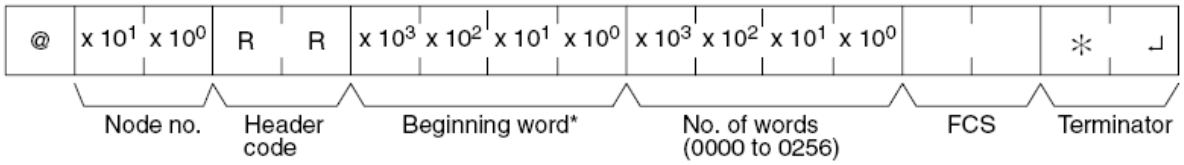
Tipo	Código	Nome	Função
I/O memória Leitura	RR	CIO AREA READ	Lê o número de *words* especificado com início no CIO *word* designado
	RL	LR AREA READ	Lê o número de *words* especificado com início no LR *word* designado
	RH	HR AREA READ	Lê o número de *words* especificado com início no HR *word* designado
	RC	TIMER/COUNTER PV READ	Lê o número de *words* especificado do temporizador/contador com início no PV *word* designado
	RD	DM AREA READ	Lê o número de *words* especificado do temporizador/contador com início no DM *word* designado
	RJ	AR AREA READ	Lê o número de *words* especificado do temporizador/contador com início no AR *word* designado
I/O memória Escrita	WR	CIO AREA WRITE	Escreve as *words* especificadas com início no CIO *word* designado
	WL	LR AREA WRITE	Escreve as *words* especificadas com início no LR *word* designado
	WH	HR AREA WRITE	Escreve as *words* especificadas com início no HR *word* designado
	WC	TIMER/COUNTER PV WRITE	Escreve as *words* especificadas com início no TIMER/COUNTER *word* designado
	WD	DM AREA WRITE	Escreve as *words* especificadas com início no DM *word* designado
	WJ	AR AREA WRITE	Escreve as *words* especificadas com início no AR *word* designado
CPU Unit Estado	MS	STATUS READ	Lê o estado do CPU
	SC	STATUS CHANGE	Altera o estado do CPU
	MF	ERROR READ	Lê os erros do CPU
Modelo *PLC* Leitura	MM	*PLC* MODEL READ	Lê o código do modelo da unidade CPU
Teste	TS	TEST	Retorna, simplesmente, um bloco igual à mensagem enviada pelo computador remoto.

Detalhes dos comandos

1. IR/SR AREA READ - RR

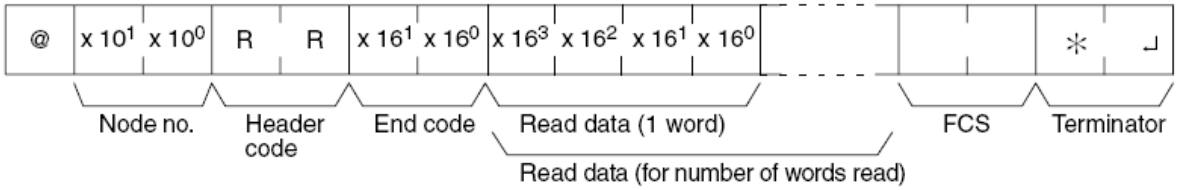
Lê o conteúdo de uma determinada *word* (IR ou SR), a começar na *word* especificada.

Formato do comando:



* *Word* de começo. Uma *word* com 0000 é devolvida se a *word* pedida não existir.

Formato da resposta:

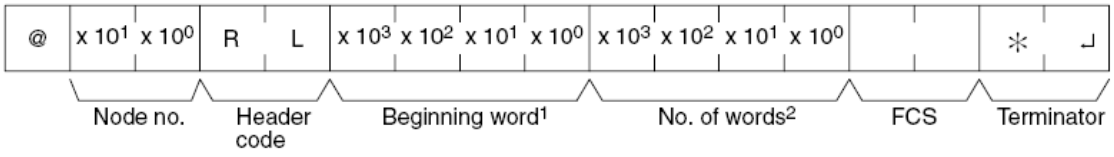


As respostas são divididas quando se lêem mais de 30 words. O conteúdo das words pedidas vem em hexadecimal.

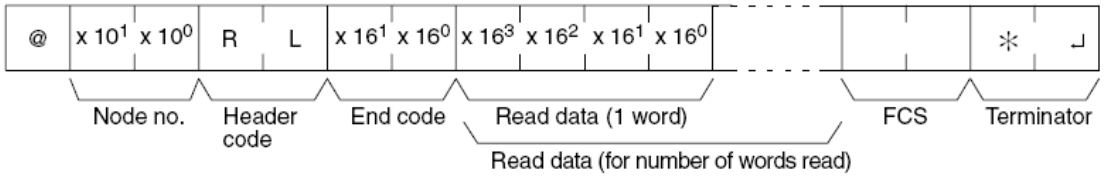
2. LR AREA READ - RL

Lê o conteúdo de uma determinada *word* LR, a começar na *word* especificada.

Formato do comando:



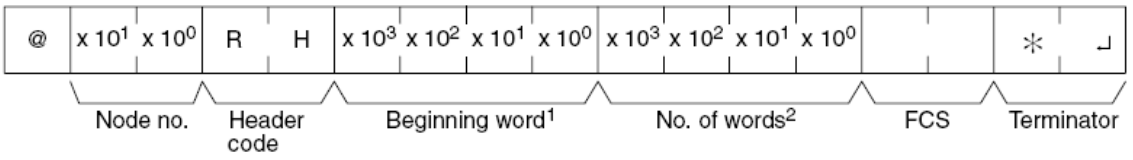
Formato da resposta:



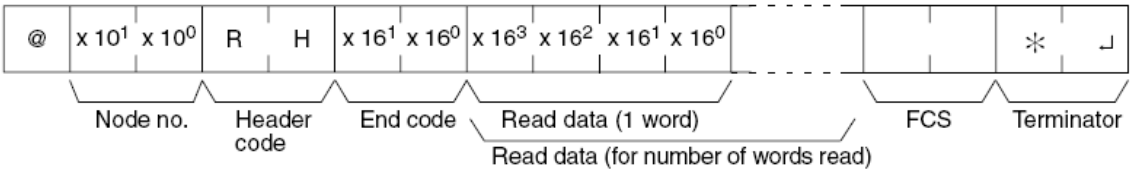
3. HR AREA READ - RH

Lê o conteúdo de uma determinada *word* HR, a começar na *word* especificada.

Formato do comando:



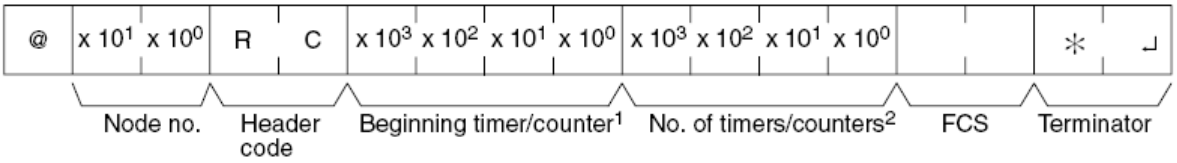
Formato da resposta:



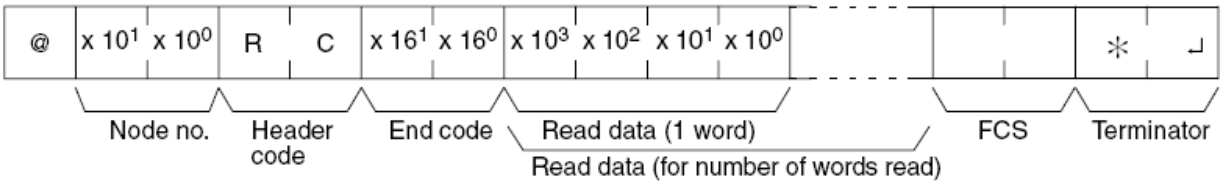
4. PV READ - RC

Lê o conteúdo de um determinado número de PV (valor presente) de timer/counter, a começar na word especificada.

Formato do comando:



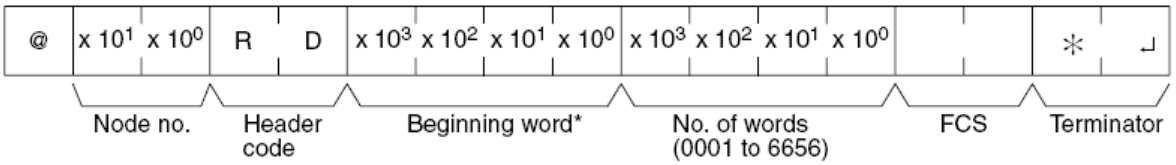
Formato da resposta:



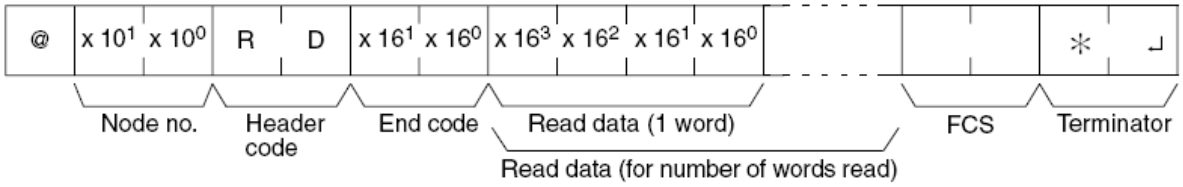
5. DM AREA READ - RD

Lê o conteúdo de uma determinada *word* DM, a começar na *word* especificada.

Formato do comando:



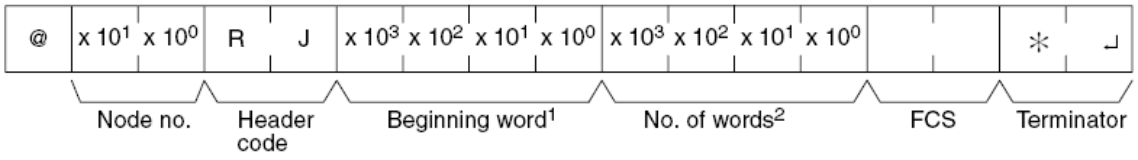
Formato da resposta:



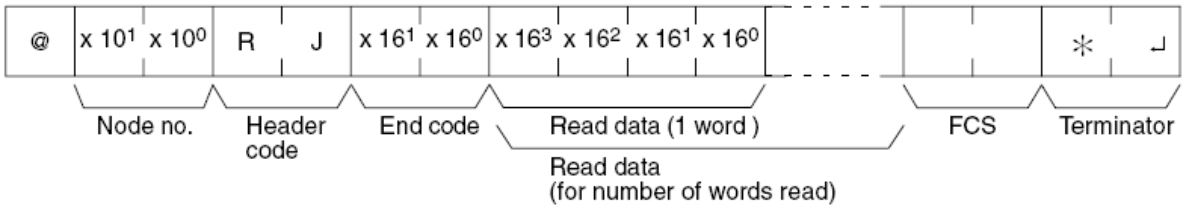
6. AR AREA READ - RJ

Lê o conteúdo de uma determinada *word* AR, a começar na *word* especificada.

Formato do comando:



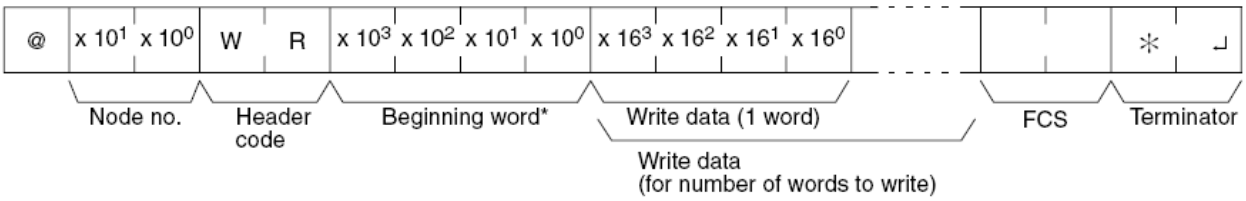
Formato da resposta:



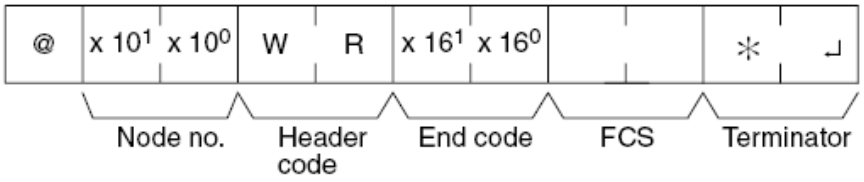
7. IR/SR AREA WRITE - WR

Escreve dados nas áreas IR e SR, a começar na *word* especificada.

Formato do comando:



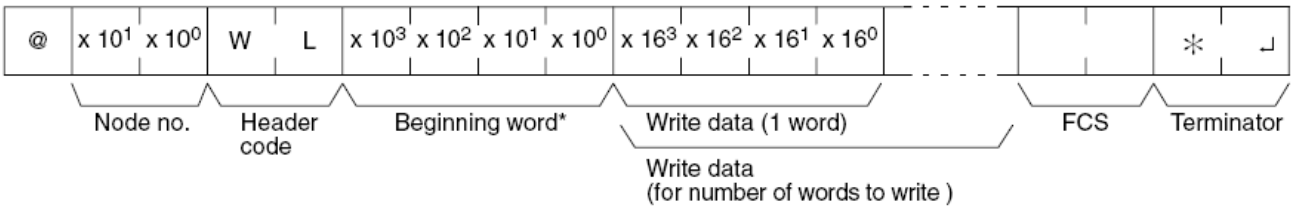
Formato da resposta:



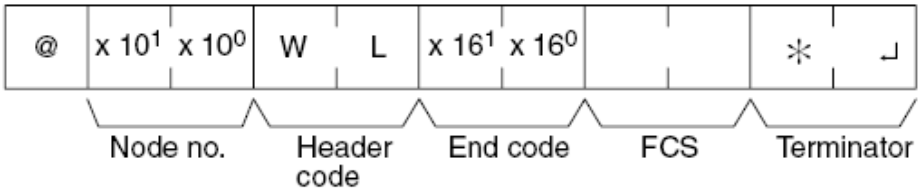
8. LR AREA WRITE - WL

Escreve dados nas áreas LR, a começar na *word* especificada.

Formato do comando:



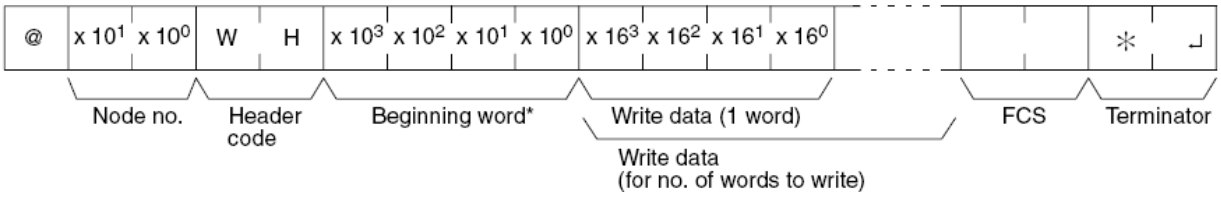
Formato da resposta:



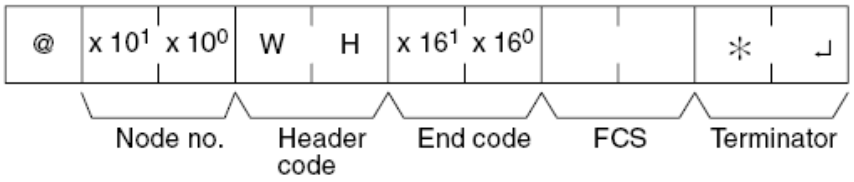
9. HR AREA WRITE - WH

Escreve dados nas áreas HR, a começar na *word* especificada.

Formato do comando:



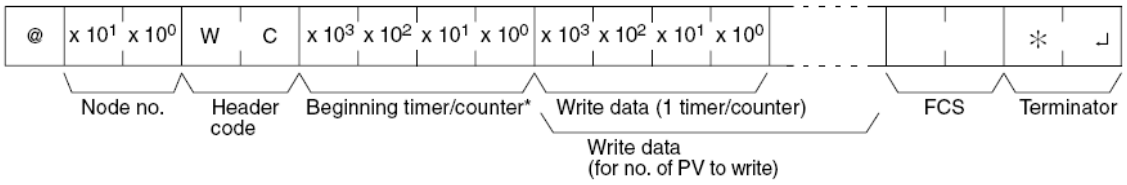
Formato da resposta:



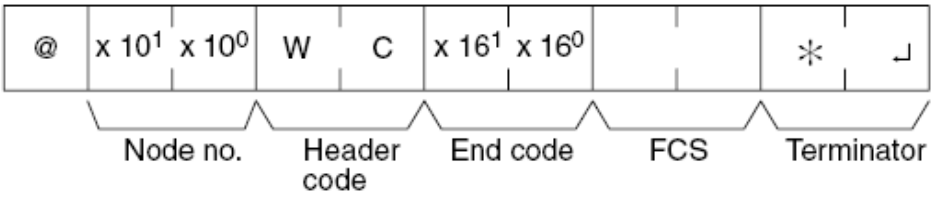
10. PV WRITE - WC

Escreve os valores presentes (PVs) dos *timer/counters*, a começar na *word* especificada.

Formato do comando:



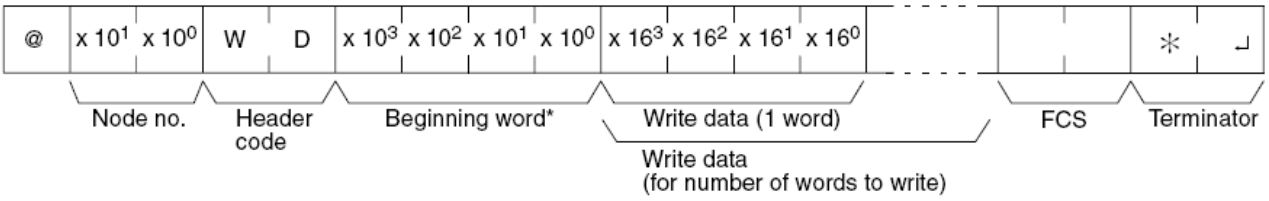
Formato da resposta:



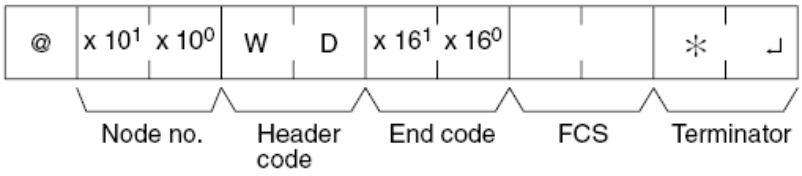
11. DM AREA WRITE - WD

Escreve dados nas áreas DM, a começar na *word* especificada.

Formato do comando:



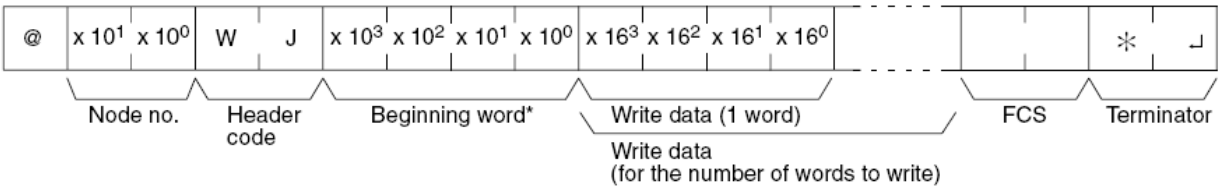
Formato da resposta:



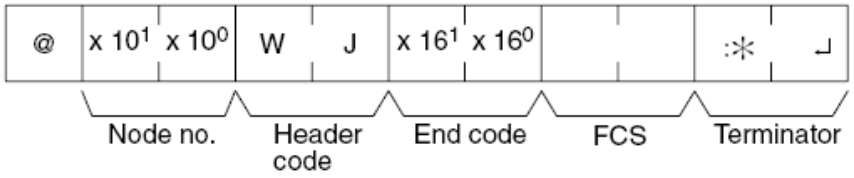
12. AR AREA WRITE - WJ

Escreve dados nas áreas AR, a começar na *word* especificada.

Formato do comando:



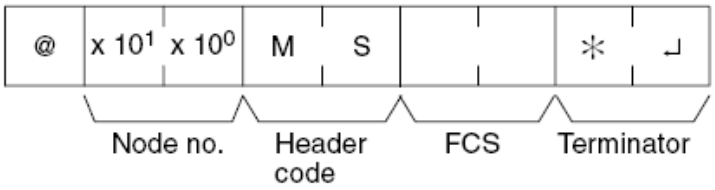
Formato da resposta:



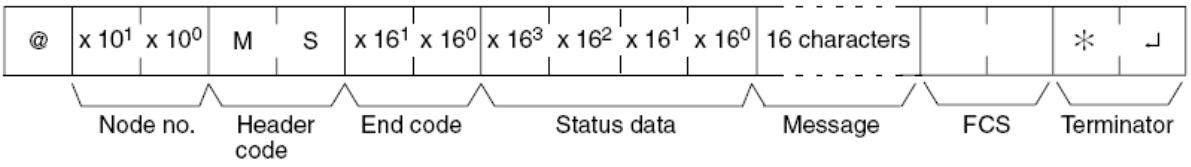
13. STATUS READ - MS

Lê o estado do CPU.

Formato do comando:



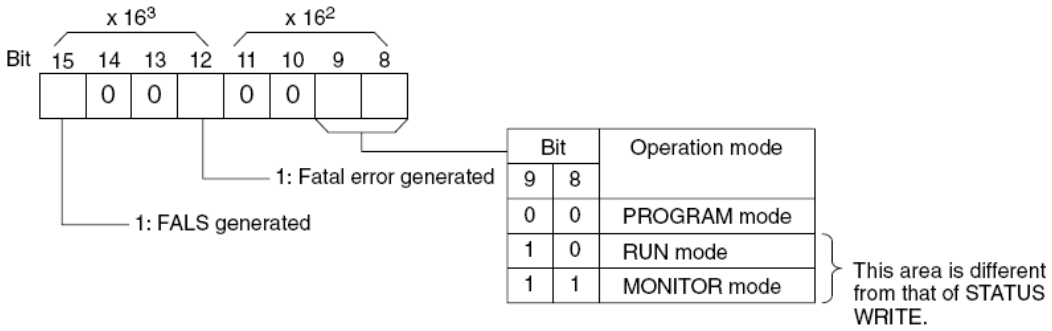
Formato da resposta:



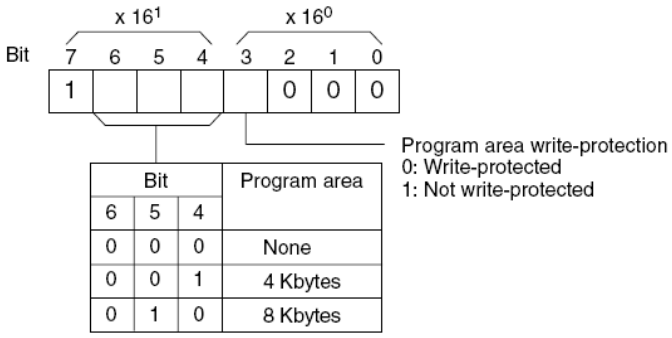
"*Status data*" consiste em quatro dígitos (dois bytes) hexadecimais. O byte mais à esquerda indica o estado do CPU e o byte mais à direita revela o tamanho da área do programa.

"*Message*" normalmente é omitida.

Byte esquerda:



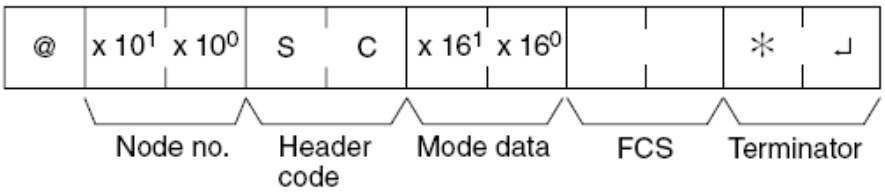
Byte direita:



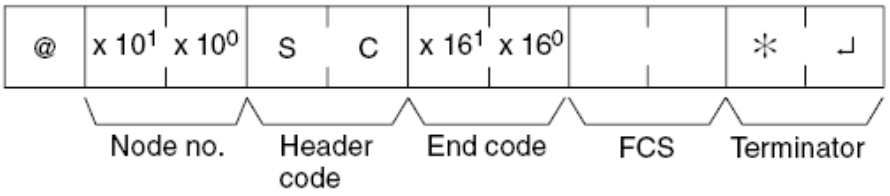
14. STATUS WRITE - SC

Muda o estado do CPU.

Formato do comando:

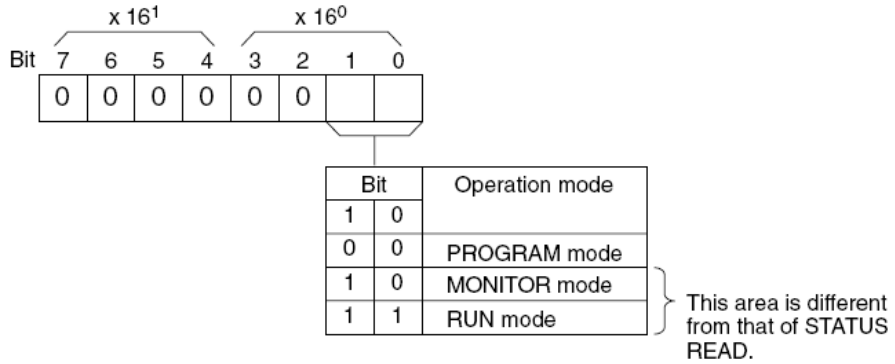


Formato da resposta:



"Mode data" consiste em 2 dígitos (um byte) hexadecimais. Com os dois bits mais à esquerda, especifica o estado do CPU. Todos os outros bits ficam a "0".

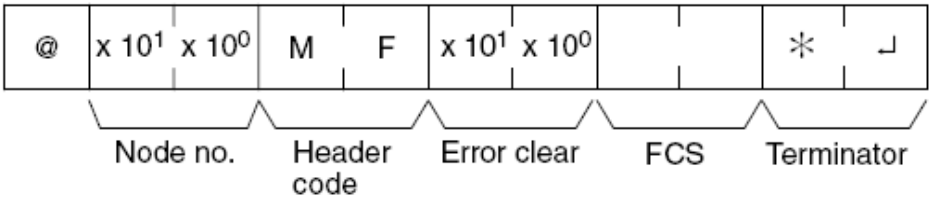
Modedata



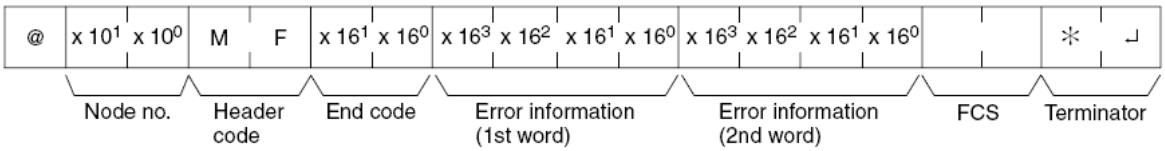
15. ERROR READ - MF

Lê e limpa os erros do CPU. Também verifica se erros anteriores foram limpos.

Formato do comando:



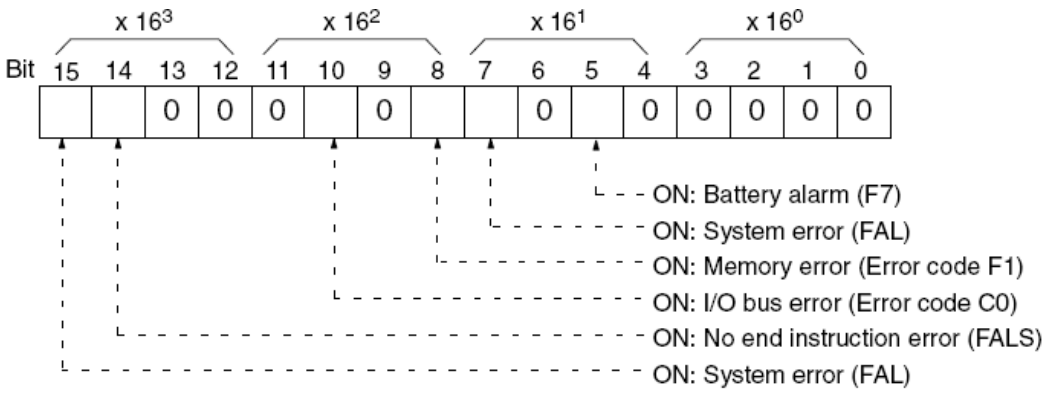
Formato da resposta:



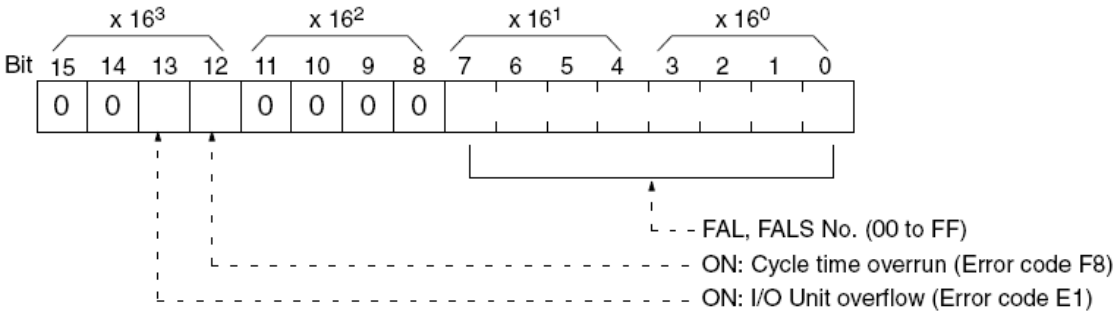
error clear "Error Clear" se for 01 limpa os erros e se for 00 não limpa os erros. Erros do tipo fatal só podem ser limpos quando o CPU está no modo Program.

"Error information" a informação do erro surge em duas words.

Primeira *word*



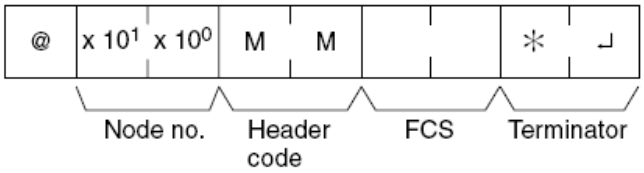
Segunda *word*



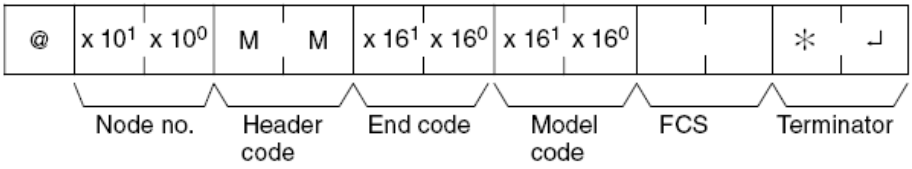
16. PC MODEL READ - MM

Lê o modelo do CPU.

Formato do comando:



Formato da resposta:



"*Model code*" indica o modelo do CPU em 2 dígitos hexadecimal:

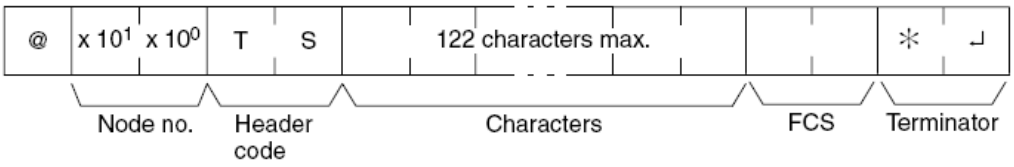
Tabela 12 - Listagem dos códigos relativos a cada modelo de PLC Omron.

Model code	Model
30	CS/CJ
01	C250
02	C500
03	C120/C50
09	C250F
0A	C500F
0B	C120F
0E	C2000
10	C1000H
11	C2000H/CQM1/CPM1
12	C20H/C28H/C40H, C200H, C200HS, C200HX/HG/HE (-ZE)
20	CV500
21	CV1000
22	CV2000
40	CVM1-CPU01-E
41	CVM1-CPU11-E
42	CVM1-CPU21-E

17. TEST - TS

Retorna, inalterado, o bloco da mensagem enviado pelo computador remoto.

Formato do comando:



Formato do resposta:

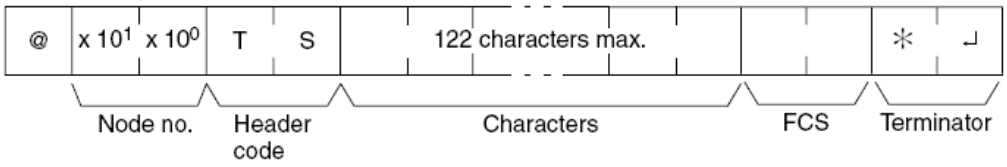


Tabela 13 - Listagem dos possíveis erros ao comunicar com PLC Omron.

End code	Conteúdo	Causa provável	Medidas para corrigir
00	Compilação normal	Não existe problema	
01	Não executável no modo RUN	O comando enviado não pode ser executado enquanto o *PLC* está no modo RUN	Ver a relação entre o comando e estado do *PLC*
02	Não executável no modo MONITOR	O comando enviado não pode ser executado enquanto o *PLC* está no modo RUN	
03	UM protecção anti escrita	O UM do *PLC* está em protecção anti-escrita	Coloque a OFF o pin 1 do SW1
04			Verifique o programa
0B	Não executável no modo PROGRAM	O comando enviado não pode ser executado enquanto o *PLC* está no modo RUN	
13	Erro FCS	O FCS da mensagem recebida está incorrecto	Verificar o modo de calcular o FCS. Se não houver influência do ruído, transferir novamente o comando.
14	Erro de formato	O formato do comando está incorrecto, ou o comando foi dividido sem o poder ser.	Ver o formato e transferir novamente o comando.
15	Erro nos dados	Os dados estão fora do alcance previsto.	Corrigir os dados e transferir o comando novamente.
16	Comando não suportado	A operação especificada em uma leitura ou escrita no comando SV não existe	Verificar os dados de início do programa

18	Erro no Comprimento da *frame*	O comprimento máximo da *frame* de 131 bytes foi excedido.	Verifique o comando e divida-o em *frames* múltiplas se necessário.
19	Não executável		
20	Não se pode criar a tabela		
21	Não executável devido à unidade de CPU		
23	Memória protegida		
A3	Abortado devido a um erro FCS na transmissão de dados		Corrija os dados de comando e volte a enviar a mensagem
A4	Abortado devido a um erro de formato na transmissão de dados		
A5	Abortado devido ao número de dados de entrada na transmissão de dados		
A8	Abortado devido a um erro no comprimento da mensagem na transmissão de dados		

Biblioteca de ligação dinâmica (DLL)

O presente texto tem como objectivo dar a conhecer o código implementado em linguagem Visual Basic, código este, que permite a interacção de um PC remoto com um autómato da *Omron*. A DLL permite, através de várias funções, ler áreas de memória do autómato, escrever nas mesmas áreas, ler o estado do autómato e altera-lo, ler o modelo do autómato ou até mesmo ler os erros.

Funções do objecto *Host* da DLL em Visual Basic:

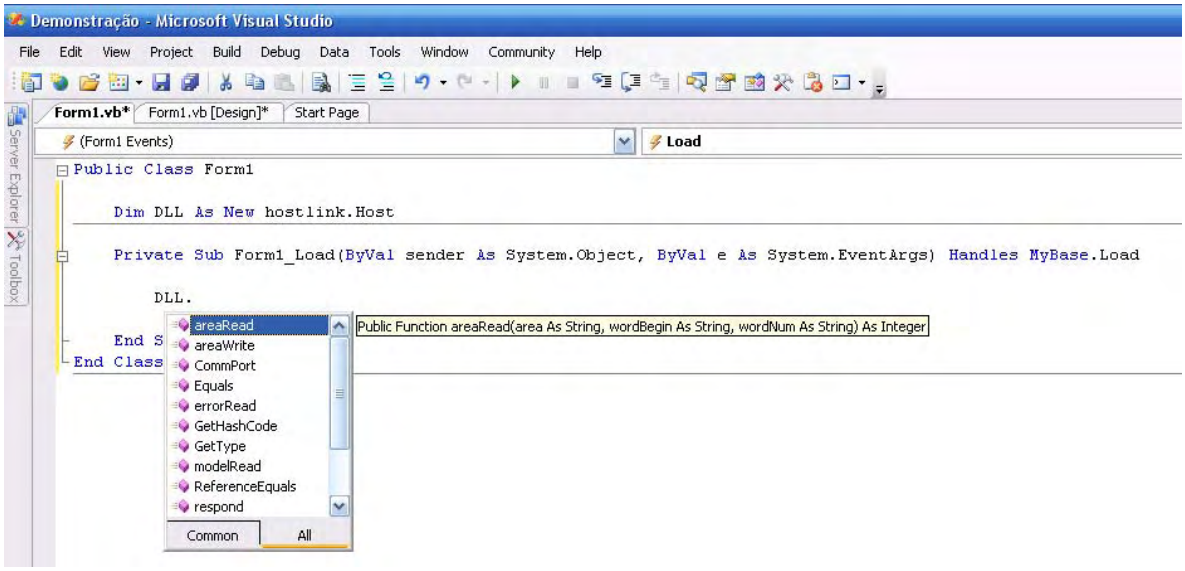


Figura 54 – Ilustração da DLL em VB

Função Area Read

A presente função é uma função pública e envia para o autómato o comando que lhe pede uma mensagem de resposta, na qual vem o valor das áreas que se pretendem ler. Fazem parte dos parâmetros de entrada da função o código do comando, a word de início para a leitura (em decimal) e também o número de words que se pretende ler (também em decimal). A função devolve um "0" se o comando for enviado correctamente, um "1" se houver uma falha na comunicação ou um "2" se a mensagem for muito longa ou o comando pretendido não for um caso possível.

```
Public Function areaRead(ByVal area As String, ByVal wordBegin As String, ByVal
wordNum As String) As Integer

        Dim msg, A, S As String
        Dim L As Integer

        A = area

        If (A = "RR" Or A = "RL" Or A = "RH" Or A = "RC" Or A = "RD" Or A = "RJ"
Or A = "RE") Then

            msg = "@00" & area & wordBegin & wordNum 'A parte inicial da mensagem
é concatenada.

        Else

            msg = "ERRO"

        End If
```

```
        L = Len(msg) 'Medido o número de caracteres da mensagem para fazer o FCS

        If (L < 128 Or msg <> "ERRO") Then 'A mensagem não enviada para o PLC se
a mensagem contiver mais de 128 caracteres ou se a área requisitada pelo
utilizador não pertencer a esta function.

            S = FCS(msg, 1) 'Chama a função FCS

            Try

                serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio
para o autómato

                Return 0 'Se tudo correr bem é devolvido o inteiro 0

            Catch ex As Exception

                Return 1 'Caso haja uma falha na comunicação no envio da mensagem
é enviado o inteiro 1

            End Try

        Else

            Return 2 'Caso a mensagem seja muito longa ou o comando pretendido
não seja um caso possível

        End If

   End Function
```

Função Area Write

A presente função é uma função pública e envia para o autómato o valor com o qual se pretende ocupar as memórias. Fazem parte dos parâmetros de entrada da função o código do comando, a *word* de início da escrita (em decimal) e também as *words* que se pretendem escrever (em hexadecimal). A função devolve um "0" se o comando for enviado correctamente, um "1" se houver uma falha na comunicação. A funçao envia um "2" se a mensagem for muito longa ou o comando pretendido não for um caso possível.

```
Public Function areaWrite(ByVal area As String, ByVal wordBegin As String, ByVal
writeData As String) As Integer

        Dim msg, A, S As String
        Dim L As Integer

        A = area

        If (A = "WR" Or A = "WL" Or A = "WH" Or A = "WC" Or A = "WD" Or A = "WJ"
Or A = "WE") Then
```

```
            msg = "@00" & area & wordBegin & writeData 'A parte inicial da
mensagem é concatenada.

        Else

            msg = "ERRO"

        End If

        L = Len(msg) 'Medido o número de caracteres da mensagem

        If (L < 128 Or msg <> "ERRO") Then 'Antes de ser enviado para o PLC se a
mensagem contiver mais de 128 caracteres ou se a área requisitada pelo utilizador
não pertencer a esta função ela não é enviada.

            S = FCS(msg, 1)

            Try
                serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio
para o autómato

                Return 0 'Se tudo correr bem é devolvido o inteiro 0

            Catch ex As Exception

                Return 1 'Caso haja uma falha na comunicação ou no envio da
mensagem é enviado o inteiro 1

            End Try

        Else

            Return 2 'Caso a mensagem seja muito longa ou o comando pretendido
não seja um caso possível

        End If

    End Function
```

Função Status Read

Esta função permite enviar para o autómato o pedido para saber qual o estado de operação. Neste caso não há parâmetros de entrada e a função devolve "0" se o envio for bem sucedido ou "1" se ocorrer uma falha de comunicação.

```
Public Function statusRead() As Integer

        Dim msg, S As String

        msg = "@00MS"

        S = FCS(msg, 1)
```

```
        Try
            serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio para o
autómato

            Return 0 'Se tudo correr bem é devolvido o inteiro 0

        Catch ex As Exception

            Return 1 'Caso haja uma falha na comunicação ou no envio da mensagem
é enviado o inteiro 1

        End Try

    End Function
```

Função Status Change

A função Status Change envia um comando para o autómato que altera o estado de operação do mesmo. Como parâmetros de entrada da função temos o ModeData, um byte com a informação relativa ao estado de operação do autómato. A função devolve um "0" se o pedido for enviado correctamente ou um "1" se houver uma falha na comunicação.

```
Public Function statusChange(ByVal Modedata As String) As Integer

        Dim msg, S As String

        msg = "@00SC" & Modedata

        S = FCS(msg, 1)

        Try
            serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio para o
autómato

            Return 0 'Se tudo correr bem é devolvido o inteiro 0

        Catch ex As Exception

            Return 1 'Caso haja uma falha na comunicação no envio da mensagem é
enviado o inteiro 1

        End Try

    End Function
```

Função Error Read

A função envia para o autómato um pedido para limpar os erros ou simplesmente para receber os índices dos erros registados. Tem como parâmetro de entrada o *error clear*. A função devolve um "0" se o comando for enviado correctamente ou um "1" se ocorrer alguma falha na comunicação.

```
Public Function errorRead(ByVal errorClear As String) As Integer

        Dim msg, S As String

        msg = "@00MF" & errorClear

        S = FCS(msg, 1)

        Try
            serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio para o
automato

            Return 0 'Se tudo correr bem é devolvido o inteiro 0

        Catch ex As Exception

            Return 1 'Caso haja uma falha na comunicação ou no envio da mensagem
é enviado o inteiro 1

        End Try

    End Function
```

Função Model Read

A função Model Read envia uma mensagem para o autómato pedindo o modelo. A função devolve "0" se o envio for efectuado com sucesso ou "1" se houver uma falha na comunicação.

```
Public Function modelRead() As Integer

        Dim msg, S As String

        msg = "@00MM"

        S = FCS(msg, 1)

        Try
            serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio para o
automato

            Return 0 'Se tudo correr bem é devolvido o inteiro 0

        Catch ex As Exception
```

```
            Return 1 'Caso haja uma falha na comunicação ou no envio da mensagem
é enviado o inteiro 1

        End Try

    End Function
```

Função Test

A seguinte função é útil nos momentos em que for necessário saber até que ponto existe uma comunicação ou não. Envia uma mensagem que o autómato devolve sem qualquer alteração. A função devolve "0" se o envio for efectuado com sucesso ou "1" se houver uma falha na comunicação.

```
Public Function Test() As Integer

        Dim msg, S As String

        msg = "@00TS11"

        S = FCS(msg, 1)

        Try
            serial_port.Write(msg & S & "*" & Chr(13)) 'Tentativa de envio para o
automato

            Return 0 'Se tudo correr bem é devolvido o inteiro 0

        Catch ex As Exception

            Return 1 'Caso haja uma falha na comunicação ou no envio da mensagem
é enviado o inteiro 1

        End Try

    End Function
```

Função Respond

A função respond tem como propósito devolver a resposta do PLC a uma outra função executada anteriormente. A função lê o buffer da porta de comunicação onde fica armazena a resposta do PLC, portanto a seguir a uma função, tal como um areaRead, deve seguir-se a respond para obter a resposta.

A função pode devolver a resposta em formato binário, decimal ou hexadecimal.

```
Public Function respond(ByVal modo As Integer) As String

        readbuffer()
```

```
        Dim bin, hexa, A As String

        A = Mid(inmsg, 4, 2)

        msg = inmsg

        inmsg = ""

        If A = "RR" Or A = "RL" Or A = "RH" Or A = "RC" Or A = "RD" Or A = "RJ"
Or A = "RE" Then 'Num comando do tipo read a variável modo é necessária para o
tratamento da mensagem de resposta

            If modo = 1 Then 'Caso o utilizador requisite as words pedidas em
binário
                bin = CalcBin(msg)
                Return msgsendstate & "$" & msgreceivedstate & "$" & bin

            ElseIf modo = 2 Then 'Caso o utilizador requisite as words pedidas em
hexadecimal
                hexa = CalcHex(msg)
                Return msgsendstate & "$" & msgreceivedstate & "$" & hexa
            Else
                Return "2" 'Erro na variável modo pedida

            End If

        ElseIf A = "WR" Or A = "WL" Or A = "WH" Or A = "WC" Or A = "WD" Or A =
"WJ" Or A = "WE" Then 'Caso o comando pedido seja do tipo write não é necessário
qualquer tratamento na resposta. A variável modo é ignorada

            Return msgsendstate & "$" & msgreceivedstate

        ElseIf A = "MS" Or A = "SC" Then

            Return msgsendstate & "$" & msgreceivedstate

        ElseIf A = "MF" Then

            If modo = 1 Then 'Caso o utilizador requisite as words pedidas em
binário
                bin = CalcBin(msg)
                Return msgsendstate & "$" & msgreceivedstate & "$" & bin

            ElseIf modo = 2 Then 'Caso o utilizador requisite as words pedidas em
hexadecimal
                hexa = CalcHex(msg)
                Return msgsendstate & "$" & msgreceivedstate & "$" & hexa
            Else
                Return "2" 'Erro na variável modo pedida

            End If

        ElseIf A = "MM" Then
```

```
            Return msgsendstate & "$" & msgreceivedstate & "$" & Mid(msg, 8, 2)
        ElseIf A = "TS" Then
            Return msgreceivedstate
        Else
            Return "3"
        End If
    End Function
```

Protocolo DNC – 50 - Fagor

Protocolo para comunicação entre um CN-8050 e um computador via DNC, sem ter em conta o conteúdo da informação que se quer transmitir. É portanto um sistema de enlace dos dados que assegura a integridade dos mesmos.

Características internas:

- Comunicação assíncrona;
- Protocolo orientado por caracteres: Tem uma série de caracteres especiais que não se podem transmitir como dados, para além de ter funções de controlo. Não é necessário usar o controlo de dados, RTS DTR CTS e DSR de RS-232.
- Potente sistema de detecção de erros de transmissão. Cada pacote de dados finaliza, com caracteres de verificação, segundo o CRC-16, que tem uma taxa de erro não detectado de 1 em 1 bilião;
- Recuperação automática de erros.

  Em linhas submetidas a interferências electromagnéticas, o protocolo retransmite automaticamente os pacotes que têm erros sem necessidade de recomeçar a comunicação.

Definições prévias:

Comunicação – transferência completa de uma informação entre os elementos que enlaçam a DNC;

-Envio de informação: enviar uma informação ao outro elemento, sendo apenas necessário o envio de uma mensagem;

-Pedido de informação: solicitar uma informação ao outro elemento e receber a informação de resposta que o escravo envia. Significa a transferência de duas mensagens, uma do cliente para o servidor, indicando o pedido, e outra do servidor para o cliente, com a informação solicitada.

Cliente – o elemento, CNC ou computador, que começa uma comunicação;

Servidor – o oposto do cliente;

Mensagem – qualquer informação simples que passa do computador para o CNC ou vice-versa. Exemplos de mensagens: pedido de um programa peça, envio de um programa peça, envio de uma ordem de controlo remoto, etc.

Mensagem de origem – mensagem enviada do cliente ao servidor. A mensagem de origem pode ser de dois tipos, envio de informação ou pedido de informação. O primeiro caracter depois do SOH indica se é pedido de informação (R) ou envio de informação (S).

Mensagem de resposta – mensagem gerada pelo servidor quando a de origem é um pedido de informação. Quando o servidor interrompe a transmissão da mensagem de origem, envia ao cliente uma mensagem de resposta informando a causa da ruptura.

Mestre – elemento que envia uma mensagem;

Escravo – o oposto do mestre;

Pacote – Quando uma mensagem é demasiado grande (mais de 1024 caracteres) divide-se em pacotes. Portanto, uma mensagem pode estar compreendida em um ou mais pacotes.

Caracteres especiais:

O protocolo DNC usa vários caracteres ASCII, que não representam nenhuma letra, número ou outro caracter de um teclado normal. Para poder representar os caracteres ASCII usam-se abreviaturas:

(SOH, STX, ETB, ETX, NAK, EOT)

ENQ – Enquiry Character – ASCII 5
Inicia o protocolo DNC, podendo também ser usado para pedir um novo envio da última mensagem.

CAN – Cancel – ASCII 24
Cancelar. Indica que se finalizou prematuramente a transferência de dados, quando o servidor interrompe a recepção de uma mensagem de origem.

0 e 1 – ASCII 48 e 49

Usam-se como reconhecimentos positivos alternados dos pacotes.

CRC – 16 – Cyclic Redundancy Check

Caracteres de verificação segundo o algoritmo CRC – 16. São quatro caracteres que se transmitem imediatamente depois do caracter de fim de pacote quer seja o ETB ou o ETX.

O resultado do CRC – 16 são 16 bits, dai o número hexadecimal de 4 dígitos, que se transmite com quatro caracteres ASCII destes 4 dígitos hexadecimais. A transformação de hexadecimal para ASCII faz-se somando 48 ao dígito hexadecimal e seguindo a correspondência segundo a tabela ASCII. Transmite-se primeiro o dígito menos significativo. O cálculo do CRC-16 começa no primeiro caracter depois de SOH ou STX e inclui o ETX ou ETB.

Transmissão de uma mensagem

Mestre	Escravo
	ENQ
0	
	SOH *S, ordem* ETB CRC-16
1	
	STX *texto* ETB CRC-16
0	
.	.
.	.
.	.
	STX *texto* STX CRC-16
0 ou 1	
	EOT

1. Fase de conexão

O elemento que quer ser mestre transmite o caracter ENQ. Se o segundo elemento estiver preparado envia o caracter 0. A partir deste momento o primeiro elemento é o mestre e o segundo é o escravo.

2. Pacote de cabeçalho

O mestre transmite um pacote de cabeçalho, no qual informa a ordem que quer levar a cabo.

O escravo decifra o CRC-16 e o significado da ordem e, se tudo tiver correcto, transmite o caracter 1.

3. Pacote de texto

Se a ordem em curso necessitar da transmissão de texto, tal como linhas de programas ou de tabelas, são transmitidos sucessivos pacotes de texto.

O escravo decifrará o CRC-16 e se não houver outros problemas, tal como saturação da memória, reconhecerá cada um dos pacotes de testo e envia 0 e 1 alternadamente.

4. Fase de desconexão

O último pacote acaba em ETX e, depois de reconhecido pelo escravo, o mestre transmite o caracter EOT como final da transmissão da mensagem.

Comunicações

O termo comunicação já foi definido no início do texto. Uma comunicação de envio de informação só requer a transmissão de uma mensagem, enquanto uma comunicação de pedido de informação exige o intercâmbio de mensagem de acordo com o seguinte esquema:

Cliente	**Servidor**
Pedido	
Mestre	**Escravo**
ENQ	
	0
SOH *R, ordem,* ETB CRC-16	
	1
EOT	
Resposta	
Escravo	**Mestre**
	ENQ
0	
	SOH *S, ordem* ETB CRC-16
1	
	STX *texto* ETB CRC-16
0	
.	.
.	.
.	.
	STX *texto* STX CRC-16
0 ou 1	

	EOT

Observações:

1. O primeiro caracter, à frente do SOH, no pacote de cabeçalho da mensagem de origem é um R, indicando que é uma mensagem de pedido de informação. Na mensagem de resposta vem um S, indicando que é uma mensagem de envio de informação.

2. A ordem de ambas as mensagens é idêntica.

Exemplos de utilização do protocolo DNC50

Pedir programa ao CN

PC	***CN***
<ENQ>	
	0
<SOH>R,PRG,TRN,000010,<ETX>CRC16	
	1
<EOT>	
	<ENQ>
0	
	<SOH>S,PRG,TRN,000010,D,DEMONSTRACAO, MX,<ETB>CRC16
1	
	<STX>PROGRAMA<ETX>CRC16
0	
	<EOT>

Enviar programa para o CN

PC	***CN***
<ENQ>	
	0
<SOH>S,PRG,TRN,000008,D,TESTE,-MX,<ETB>CRC16	
	1
<STX>PROGRAMA<ETX>CRC16	
	0
<EOT>	

Ao enviar ou receber Programas cujo conteúdo com os caracteres e CRC16 ultrapassar 1024 Kbytes, o programa é dividido em blocos e o caracter no final de cada bloco será o ETB, excepto no último bloco que é o ETX, entre blocos o receptor deverá responder 0 e 1 alternadamente.

PEDIR DIRECTÓRIO dos PROGRAMAS ao CN

PC	***CN***
<ENQ>	
	0
<SOH>R,PRG,DIR,P,0,<ETX>CRC16	
	1
<EOT>	
	<ENQ>
0	
	<SOH>S,PRG,DIR,P,0,<ETB>
1	
	<STX>M 12DIRECTÓRIOS<ETX>CRC16
0	
	<EOT>

Quando o CN envia o Directório dos Programas, seguem apenas 12 de cada vez a começar no número indicado. Para pedir os seguintes é necessário fazer novo pedido a começar na posição seguinte, ou seja, se tiver pedido a partir do 0 no pedido seguinte será a partir do 12.

Ver Estado de VARIÁVEIS

Variáveis do *PLC*

PC	***CN***
<ENQ>	
	0
<SOH>R,*PLC*,M200,<ETX>CRC16	
	1
<EOT>	
	<ENQ>
0	
	<SOH>S,*PLC*,M200=0,<ETX>CRC16
1	
	<EOT>

Variáveis do CN

PC	***CN***
<ENQ>	
	0
<SOH>R,VAR,TOOL,<ETX>CRC16	
	1
<EOT>	
	<ENQ>
0	
	<SOH>S,VAR,TOOL, 0,<ETX>CRC16
1	
	<EOT>

Alterar estado de variável do *PLC*

PC	***CN***
<ENQ>	
	0
<SOH>S,*PLC*,M200=1,<ETX>CRC16	
	1
<EOT>	

Descrição dos caracteres

Caracter	Nome	Descrição
SOH	Inicio do cabeçalho	SOH identifica o começo do cabeçalho da transferência de dados.
STX	Inicio do texto	Identifica o início de um bloco de programa.
ETB	Fim do texto	Termina um Bloco de dados.
DC1	Inicio da transferência de dados (XON)	Inicia a transferência de dados
DC3	Fim da transferência de dados (XOFF)	Termina a transferência de dados
ETX	Fim de texto	É transmitido no final de um programa
EOT	Fim de transmissão	Termina a transferência de dados e estabelece o estado inactivo.
ACK	Acordo	É transmitido pelo receptor quando um bloco de dados é transferido sem erros.
NAK	Acordo negativo	É transmitido pelo receptor quando um bloco de dados é transferido com erro. O transmissor tem de reenviar o bloco de dados.

A4 - Equipamentos testados

Siemens S7-300

A família Simatic S7-300 tem sido empregada com sucesso nos mais diversos equipamentos industriais, com destaque na construção de máquinas de série ou especiais para produção ou processo. Esta série permite o uso de E/S digitais ou analógicas e, por meio da porta Profibus-DP integrada, a CPU pode ser usado como mestre ou escravo de uma rede Profibus-DP. Quando mestre pode controlar até 64 dispositivos escravos, ou estações de E/S distribuídas". A comunicação com outros controladores, interfaces homem máquina (IHM), ou dispositivos de programação *Siemens* pode ser implementada através da interface MPI, também integrada com o CPU.

O endereçamento de bits processa-se por AX.Y, sendo A a área a que se pretende aceder, X o Byte e Y o bit. Se pretender aceder em formato de byte será ABX, sendo A a área e X o byte a aceder; se for em formato de Word,é AWX e em Double Word fica ADX, mantendo-se a denominação para A e X.

Descrição das variáveis:

Entradas Digitais: Ix,y

Descrição: definida pela letra I; x – Posição da carta na rack; y – Entrada na carta.

Saídas Digitais: Qx,y

Descrição: definida pela letra Q; x – Posição da carta na rack; y – Entrada na carta.

Bit´s Internos: M, (M0.0 … M255.7)

Descrição: identificação e endereçamento efectuados em decimal.

Byte, 8 bit's: MB, (MB0 … MB255)

Descrição: identificação efectuada em decimal.

Word, 16 bit's: MW, (MW0 … MW254)

Descrição: identificação efectuada em decimal.

Double Word, 32 bit's: MD, (MD0 … MD252)

Descrição: identificação efectuada em decimal.

Temporizadores: (T0 … T255)

Descrição: identificação efectuada em decimal.

Modo de definir o tempo de contagem: s5t#_H__M__S__MS

- Ex: 1min e 5seg : s5t#1M5S

Contadores: (Z0 … Z255)

Descrição: identificação efectuada em decimal.

Siemens S7-200

Os autómatos *Siemens* Simatic S7 200 (Figura 27) são do tipo compacto, incorporam a unidade central de processamento (CPU), a fonte de alimentação e as entradas e saídas digitais. Existem vários e diferentes CPU's, aos quais se podem ligar distintos módulos de expansão.

Os autómatos S7 200 armazenam a informação em diferentes áreas de memória que por sua vez se encontra organizada por áreas, de acordo com as funções a realizar – conjunto de bits.

O endereçamento de bits é feito da mesma forma que para o S7 300.

Área das Entradas/ Saídas (I/Q): área que corresponde aos bits de entrada/saída (I/O) do autómato, sendo que os primeiros bits I/O correspondem aos terminais das entradas e saídas físicas do autómato e são as imagens lógicas do estado eléctrico desses terminais. Os restantes bits são apenas acessíveis através da programação.

Entradas: I

Saídas: Q

Área das variáveis (V): área utilizada para guardar resultados/estados intermédios realizados pelo programa. Estas variáveis são também conhecidas por variáveis de trabalho, sendo as mais comuns para elaborar os programas.

Área das Marcas (M): área onde é possível guardar resultados intermédios de operações e outras informações de controlo.

Área das marcas especiais (SM): disponibiliza sinais de relógio, flags, bits de controlo, informação sobre o estado do *PLC* e outras. Exemplo: SM0.0 está sempre activo quando o *PLC* está em modo RUN.

Área dos temporizadores (T): associada aos temporizadores.

Área dos contadores (C): associada aos contadores.

Mitsubishi

O FX2N tem muitas das propriedades que normalmente são encontradas em controladores distintamente maiores, tal como a capacidade de lidar com Dword (32-bits), para além de um vasto leque de opções de configuração a nível da comunicação.

O FX2N é um dos mais rápidos *PLC* compactos, com uma grande capacidade de comunicação e expansão, sendo re sublinhar que existem módulos de funções especiais para configuração de sistemas especificamente personalizados.

Principais propriedades:

- Entre 16 a 256 entradas e saídas
- Várias opções de comunicação (Profibus-DP, CC-Link, DeviceNet e AS-interface)

Variáveis mais comuns nos *PLC's* da *Mitsubishi*:

Entradas digitais: X – ex. X2 – o endereçamento é efectuado em decimal.

Saídas digitais: Y – ex. Y3 – o endereçamento é efectuado em decimal.

Bit´s internos: M – ex. M10 – varáveis de memória, a identificação e endereçamento é realizado em decimal.

Variáveis Internas: D – ex. D20 – Word's ou Double Word's (16 ou 32 Bit's) para processamento de números ou palavras. O endereçamento em decimal.

Temporizadores - variáveis necessárias para a implementação de um temporizador. O endereçamento especificado em decimal.

- TC – Identificação do temporizador (0..255).
- TN – Indicação da contagem actual do temporizador.
- TS – Variável de saída, indica o fim de contagem do temporizador.

Contadores: CC; CN; CS – variáveis necessárias para a realização e implementação de um contador. As variáveis utilizadas são idênticas às descritas anteriormente.

Omron

A Omron é uma gama de autómatos do tipo compacto, com saídas a relé ou transístor, apresentando o modelo CPM1A.

A memória do autómato é constituída pela memória de programas e pela memória de dados. Enquanto a memória de programas armazena as instruções que constituem o programa, a memória de dados guarda as variáveis de entrada, variáveis de saída, valores de cálculos intermédios e outros.

Bits internos (IR) - Bits que podem memorizar o estado de variáveis ou o resultado de equações, durante a execução do programa. São formados por:

Bits de trabalho (Work Bits) - Bits que podem ser utilizados para qualquer função no programa, excepto como bits de E/S.

Bits de E/S (I/O) - Bits que se destinam à programação de entradas e saídas, internas e externas. Os primeiros bits de E/S correspondem fisicamente aos terminais de entrada e saída, são as imagens lógicas dos estados eléctricos das entradas/saídas, reflectindo o estado On/Off desses terminais. Os restantes de E/S apenas são acessíveis através de programação.

Bits especiais (SR) - Bits utilizados para funções especificas. Disponibilizam sinais de relógio, *flags*, bits de controlo e informações sobre o estado do autómato.

Bits de retenção (HR) - Bits usados para guardar e memorizar dados quando o autómato é desligado. Estes bits utilizam-se da mesma forma que os bits de trabalho.

Bits de ligação (LR) - Bits usados na comunicação com outros autómatos ou, caso esta não exista, podem ser utilizados como bits de trabalho.

Bits auxiliares (AR) - Bits com funções similares aos bits SR. Disponibilizam informação sobre a operação do *PLC*.

Bits de temporizadores/contadores (TC) - Bits que estão associados aos temporizadores/contadores. Activam-se no final das temporizações ou contagens.

Memória de dados (DM) - Memória para armazenar os dados que também contém os parâmetros de funcionamento do autómato (Setup).